

Patterns of Application Development Using AI

Obie Fernandez

Foreword by Gregor Hohpe

Leanpub

Édition Française

Patrons de Développement d'Applications Utilisant l'IA (Édition Française)

Obie Fernandez

Ce livre est en vente à

<http://leanpub.com/patterns-of-application-development-using-ai-fr>

Cette version a été publiée le 2025-01-23



Ce livre est publié par [Leanpub](#). Leanpub permet aux auteurs et aux éditeurs de bénéficier du processus Lean Publishing. [Lean Publishing](#) consiste à publier à l'aide d'outils très simples de nombreuses itérations d'un livre ebook en cours de rédaction, d'obtenir des retours et des commentaires des lecteurs afin d'améliorer le livre.

© 2025 Obie Fernandez

Tweet ce livre !

S'il vous plaît aidez Obie Fernandez en parlant de ce livre sur [Twitter](#) !

Le hashtag conseillé pour ce livre est [#poaduai](#).

Découvrez ce que les gens disent à propos du livre en cliquant sur ce lien pour rechercher ce hashtag sur Twitter :

[#poaduai](#)

À Victoria, ma reine indomptable, ma muse, ma lumière et mon amour

Aussi par Obie Fernandez

Patterns of Application Development Using AI

The Rails 8 Way

The Rails 7 Way

XML The Rails Way

Serverless

El Libro Principiante de Node

The Lean Enterprise

Table des matières

Préface par Gregor Hohpe	i
Préface	ii
À propos du livre	iii
À propos des exemples de code	iii
Ce que je ne couvre pas	iii
À qui s'adresse ce livre	iii
Construire un Vocabulaire Commun	iii
S'impliquer	iii
Remerciements	iii
Qu'en est-il des illustrations ?	iv
À propos du Lean Publishing	iv
À propos de l'auteur	v
Introduction	1
Réflexions sur l'Architecture Logicielle	2
Qu'est-ce qu'un Grand Modèle de Langage ?	3
Comprendre l'Inférence	5
Réflexion sur la Performance	28
Expérimentation avec Différents Modèles GML	30
Systèmes d'IA Composés	31

Partie 1 : Approches et Techniques Fondamentales	39
Restreindre le Chemin	40
L'espace latent : Incompréhensiblement vaste	42
Comment Le Chemin Est “Rétréci”	47
Modèles bruts versus modèles ajustés par instructions	50
Ingénierie des Prompts	57
Distillation de Prompts	74
Qu'en est-il de l'ajustement fin ?	81
Retrieval Augmented Generation (RAG)	83
Qu'est-ce que la Retrieval Augmented Generation ?	83
Comment fonctionne le RAG ?	83
Pourquoi utiliser le RAG dans vos applications ?	83
Mise en oeuvre de RAG dans votre application	83
Découpage en propositions	84
Exemples concrets de RAG	84
Optimisation Intelligente des Requêtes (IQO)	85
Reclassement	85
Évaluation RAG (RAGAs)	85
Défis et Perspectives d'Avenir	87
Multitude de travailleurs	89
Les travailleurs IA comme composants réutilisables indépendants	90
Gestion des comptes	92
Applications E-commerce	93
Applications dans le domaine de la santé	102
L'Agent IA comme Gestionnaire de Processus	106
Intégration des travailleurs IA dans l'architecture de votre application	110

TABLE DES MATIÈRES

Composabilité et orchestration des workers IA	113
Combiner le TLN traditionnel avec les LLM	122
Utilisation d'outils	126
Qu'est-ce que l'utilisation d'outils?	126
Le potentiel de l'utilisation d'outils	128
Le flux de travail de l'utilisation d'outils	129
Meilleures pratiques pour l'utilisation des outils	144
Composition et Chaînage d'Outils	149
Orientations Futures	151
Traitement de flux	153
Implémentation d'un ReplyStream	154
La «Boucle de Conversation»	160
Auto-continuation	163
Conclusion	165
Données auto-réparatrices	167
Étude de cas pratique : Réparation du JSON défectueux	170
Considérations et Contre-indications	175
Génération de Contenu Contextuel	191
Personnalisation	192
Productivité	194
Itération et expérimentation rapides	196
Localisation alimentée par l'IA	199
L'Importance des Tests Utilisateur et des Retours	201
Interface utilisateur générative	203
Génération de texte pour les interfaces utilisateur	205
Définir l'interface utilisateur générative	214

TABLE DES MATIÈRES

Exemple	216
Le passage à une conception orientée résultats	219
Défis et considérations	221
Perspectives d'avenir et opportunités	223
Orchestration intelligente des flux de travail	226
Besoin métier	227
Avantages clés	228
Modèles clés	229
Gestion et Récupération des Exceptions	231
Mise en oeuvre pratique de l'orchestration des flux de travail intelligents	234
Surveillance et journalisation	249
Considérations d'extensibilité et de performance	254
Tests et validation des flux de travail	259
Partie 2 : Les Patrons	268
Ingénierie des Prompts	269
Chaîne de réflexion	270
Mode Switch	271
Attribution de Rôle	272
Prompt Object	273
Modèle de Prompt	274
Entrées/Sorties structurées	275
Chaînage de prompts	276
Réécriture de Prompts	277
Délimitation des réponses	278
Query Analyzer	279
Réécriveur de requêtes	281
Ventriloquist	282

TABLE DES MATIÈRES

Composants Discrets	283
Prédicat	284
Façade d'API	285
Result Interpreter	287
Virtual Machine	288
Spécification et Tests	288
Human In The Loop (HITL)	290
Modèles de Haut Niveau	290
Escalade	291
Boucle de rétroaction	292
Radiation Passive d'Information	293
Prise de Décision Collaborative (PDC)	295
Apprentissage Continu	296
Considérations éthiques	296
Avancées technologiques et perspectives d'avenir	296
Gestion intelligente des erreurs	298
Approches traditionnelles de gestion des erreurs	298
Diagnostic contextuel des erreurs	299
Rapport d'erreur intelligent	300
Prévention prédictive des erreurs	301
Récupération Intelligente des Erreurs	301
Communication Personnalisée des Erreurs	302
Flux de Travail Adaptatif de Gestion des Erreurs	303
Contrôle Qualité	304
Eval	305
Garde-fou	307
Garde-fous et Évaluations : Les Deux Faces d'une Même Médaille	307

Glossaire	309
Glossaire	309
Index	314

Préface par Gregor Hohpe

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Préface

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

À propos du livre

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

À propos des exemples de code

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Ce que je ne couvre pas

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

À qui s'adresse ce livre

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Construire un Vocabulaire Commun

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

S'impliquer

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Remerciements

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Qu'en est-il des illustrations?

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

À propos du Lean Publishing

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

À propos de l'auteur

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Introduction



Si vous êtes impatient de commencer à intégrer les Grands Modèles de Langage (GML) dans vos projets de programmation, n'hésitez pas à plonger directement dans les patrons et les exemples de code présentés dans les chapitres suivants. Cependant, pour apprécier pleinement la puissance et le potentiel de ces patrons, il est utile de prendre un moment pour comprendre le contexte plus large et l'approche cohésive qu'ils représentent.

Les patrons ne sont pas simplement une collection de techniques isolées, mais plutôt un cadre unifié pour l'intégration de l'IA dans vos applications. J'utilise Ruby on Rails, mais ces patrons devraient fonctionner dans pratiquement n'importe quel autre environnement de programmation. Ils abordent un large éventail de préoccupations, de la gestion des données et l'optimisation des performances à l'expérience utilisateur et la sécurité, fournissant une boîte à outils complète pour améliorer les pratiques de programmation traditionnelles avec les capacités de l'IA.

Chaque catégorie de patrons s'attaque à un défi ou une opportunité spécifique qui survient lors de l'incorporation de composants d'IA dans votre application. En comprenant les relations et les synergies entre ces patrons, vous pouvez prendre des décisions éclairées sur où et comment appliquer l'IA le plus efficacement.

Les patrons ne sont jamais des solutions prescriptives et ne doivent pas être traités comme tels. Ils sont conçus pour être des éléments constitutifs adaptables qui doivent être ajustés aux exigences et contraintes uniques de votre propre application. L'application réussie de ces patrons (comme tous les autres dans le domaine du logiciel) repose sur une compréhension approfondie du domaine du problème, des besoins des utilisateurs et de l'architecture technique globale de votre projet.

Réflexions sur l'Architecture Logicielle

J'ai commencé à programmer dans les années 1980 et j'étais impliqué dans la scène hacker, et je n'ai jamais perdu mon état d'esprit hacker, même après être devenu développeur professionnel. Depuis le début, j'ai toujours eu un scepticisme sain quant à la valeur réelle apportée par les architectes logiciels dans leurs tours d'ivoire.

L'une des raisons pour lesquelles je suis personnellement si enthousiaste à propos des changements apportés par cette nouvelle vague puissante de technologie d'IA est son impact sur ce que nous considérons comme des décisions d'*architecture logicielle*. Elle remet en question les notions traditionnelles de ce qui constitue la "bonne" façon de concevoir et de mettre en œuvre nos projets logiciels. Elle remet également en question si l'architecture peut encore être considérée principalement comme *les parties d'un système qui sont difficiles à modifier*, puisque l'amélioration par l'IA rend plus facile que jamais de modifier n'importe quelle partie de votre projet, à tout moment.

Peut-être entrons-nous dans les années phares de l'approche "post-moderne" du génie logiciel. Dans ce contexte, post-moderne fait référence à un changement fondamental par rapport aux paradigmes traditionnels, où les développeurs étaient responsables

d'écrire et de maintenir chaque ligne de code. Au lieu de cela, elle embrasse l'idée de déléguer des tâches, telles que la manipulation de données, les algorithmes complexes, et même des parties entières de la logique d'application, à des bibliothèques tierces et des API externes. Ce changement post-moderne représente une rupture significative avec la sagesse conventionnelle de construire des applications depuis la base, et il met au défi les développeurs de repenser leur rôle dans le processus de développement.

J'ai toujours cru que les bons programmeurs n'écrivent que le code absolument nécessaire, basé sur les enseignements de Larry Wall et d'autres luminaires hacker comme lui. En minimisant la quantité de code écrit, nous pouvons avancer plus rapidement, réduire la surface d'exposition aux bugs, simplifier la maintenance et améliorer la fiabilité globale de leurs applications. Moins de code nous permet de nous concentrer sur la logique métier et l'expérience utilisateur, tout en déléguant d'autres travaux à d'autres services.

Maintenant que les systèmes alimentés par l'IA peuvent gérer des tâches qui étaient auparavant le domaine exclusif du code écrit par l'homme, nous devrions pouvoir être encore plus productifs et agiles, avec une concentration plus grande que jamais sur la création de valeur commerciale et l'expérience utilisateur.

Bien sûr, il y a des compromis à déléguer d'énormes parties de votre projet à des systèmes d'IA, comme la perte potentielle de contrôle et le besoin de mécanismes robustes de surveillance et de retour d'information. C'est pourquoi cela nécessite un nouveau ensemble de compétences et de connaissances, y compris au moins une compréhension fondamentale du fonctionnement de l'IA.

Qu'est-ce qu'un Grand Modèle de Langage ?

Les Grands Modèles de Langage (GML) sont un type de modèle d'intelligence artificielle qui ont gagné une attention significative ces dernières années, depuis le lancement de GPT-3 par OpenAI en 2020. Les GML sont conçus pour traiter, comprendre et générer

le langage humain avec une précision et une fluidité remarquables. Dans cette section, nous allons jeter un bref coup d'œil sur le fonctionnement des GML et pourquoi ils sont bien adaptés pour construire des composants de systèmes intelligents.

À leur cœur, les GML sont basés sur des algorithmes d'apprentissage profond, spécifiquement les réseaux neuronaux. Ces réseaux sont composés de nœuds interconnectés, ou neurones, qui traitent et transmettent l'information. L'architecture de choix pour les GML est souvent le modèle Transformer, qui s'est avéré très efficace dans le traitement des données séquentielles comme le texte.

Les modèles Transformer sont basés sur le mécanisme d'attention et sont principalement utilisés pour des tâches impliquant des données séquentielles, comme le traitement du langage naturel. Les Transformers traitent les données d'entrée en une seule fois plutôt que séquentiellement, ce qui leur permet de capturer plus efficacement les dépendances à long terme. Ils disposent de couches de mécanismes d'attention qui aident le modèle à se concentrer sur différentes parties des données d'entrée pour comprendre le contexte et les relations.

Le processus d'entraînement des grands modèles de langage implique d'exposer le modèle à de vastes quantités de données textuelles, comme des livres, des articles, des sites web et des dépôts de code. Pendant l'entraînement, le modèle apprend à reconnaître les motifs, les relations et les structures dans le texte. Il capture les propriétés statistiques du langage, comme les règles grammaticales, les associations de mots et les significations contextuelles.

L'une des techniques clés utilisées dans l'entraînement des grands modèles de langage est l'apprentissage non supervisé. Cela signifie que le modèle apprend à partir des données sans étiquetage ou guidance explicite. Il découvre des motifs et des représentations par lui-même en analysant la co-occurrence des mots et des phrases dans les données d'entraînement. Cela permet aux grands modèles de langage de développer une compréhension profonde du langage et de ses subtilités.

Un autre aspect important des grands modèles de langage est leur capacité à gérer

le *contexte*. Lors du traitement d'un texte, les grands modèles de langage considèrent non seulement les mots individuels mais aussi le contexte environnant. Ils prennent en compte les mots, phrases et même paragraphes précédents pour comprendre le sens et l'intention du texte. Cette compréhension contextuelle permet aux grands modèles de langage de générer des réponses cohérentes et pertinentes. L'une des principales façons d'évaluer les capacités d'un modèle de langage donné est de considérer la taille du contexte qu'ils peuvent prendre en compte pour générer des réponses.

Une fois entraînés, les grands modèles de langage peuvent être utilisés pour une large gamme de tâches linguistiques. Ils peuvent générer du texte semblable à celui d'un humain, répondre à des questions, résumer des documents, traduire des langues et même écrire du code. La polyvalence des grands modèles de langage les rend précieux pour construire des composants de systèmes intelligents qui peuvent interagir avec les utilisateurs, traiter et analyser des données textuelles, et générer des résultats significatifs.

En incorporant les grands modèles de langage dans l'architecture d'application, vous pouvez créer des composants d'IA qui comprennent et traitent les entrées utilisateur, génèrent du contenu dynamique et fournissent des recommandations ou actions intelligentes. Mais travailler avec les grands modèles de langage nécessite une considération attentive des besoins en ressources et des compromis de performance. Les grands modèles de langage sont gourmands en calcul et peuvent nécessiter une puissance de traitement et une mémoire significatives (en d'autres termes, de l'argent) pour fonctionner. La plupart d'entre nous devront évaluer les implications financières de l'intégration des grands modèles de langage dans nos applications et agir en conséquence.

Comprendre l'Inférence

L'inférence fait référence au processus par lequel un modèle génère des prédictions ou des sorties basées sur de nouvelles données non vues. C'est la phase où le modèle

entraîné est utilisé pour prendre des décisions ou générer du texte, des images ou d'autres contenus en réponse aux entrées utilisateur.

Pendant la phase d'entraînement, un modèle d'IA apprend à partir d'un large ensemble de données en ajustant ses paramètres pour minimiser l'erreur dans ses prédictions. Une fois entraîné, le modèle peut appliquer ce qu'il a appris à de nouvelles données. L'inférence est la façon dont le modèle utilise ses motifs et connaissances apprises pour générer des sorties.

Pour les grands modèles de langage, l'inférence implique de prendre une invite ou un texte d'entrée et de produire une réponse cohérente et contextuellement pertinente, sous forme de flux de *tokens* (dont nous parlerons bientôt). Cela peut être répondre à une question, compléter une phrase, générer une histoire ou traduire du texte, parmi de nombreuses autres tâches.



Contrairement à la façon dont vous et moi pensons, la "réflexion" d'un modèle d'IA via l'inférence se produit en une seule opération sans état. C'est-à-dire que sa réflexion est limitée à son processus de génération. Il doit littéralement penser à voix haute, comme si je vous posais une question et n'acceptais une réponse de vous que dans un style "flux de conscience".

Les Grands Modèles de Langage Existents en Plusieurs Tailles et Saveurs

Bien que pratiquement tous les grands modèles de langage populaires soient basés sur la même architecture transformée de base et entraînés sur d'énormes ensembles de données textuelles, ils existent en différentes tailles et sont affinés pour différents objectifs. La taille d'un grand modèle de langage, mesurée par le nombre de paramètres dans son réseau neuronal, a un impact important sur ses capacités. Les modèles plus grands avec plus de paramètres, comme GPT-4, qui selon les rumeurs dispose de 1 à 2 billions de paramètres, sont généralement plus compétents et capables que les modèles plus petits.

Cependant, les modèles plus grands nécessitent aussi beaucoup plus de puissance de calcul pour fonctionner, ce qui se traduit par des coûts plus élevés lorsque vous les utilisez via des appels API.

Pour rendre les grands modèles de langage plus pratiques et adaptés à des cas d'utilisation spécifiques, les modèles de base sont souvent affinés sur des ensembles de données plus ciblés. Par exemple, un grand modèle de langage peut être entraîné sur un large corpus de dialogues pour le spécialiser dans l'IA conversationnelle. D'autres sont [entraînés sur du code](#) pour leur inculquer des connaissances en programmation. Il existe même des modèles qui sont [spécialement entraînés pour des interactions de type jeu de rôle avec les utilisateurs](#)!

Modèles de récupération vs Modèles génératifs

Dans le monde des grands modèles de langage (LLMs), il existe deux approches principales pour générer des réponses : les modèles basés sur la récupération et les modèles génératifs. Chaque approche possède ses propres forces et faiblesses, et comprendre les différences entre elles peut vous aider à choisir le modèle le plus adapté à votre cas d'utilisation spécifique.

Modèles basés sur la récupération

Les modèles basés sur la récupération, également connus sous le nom de modèles de recherche d'information, génèrent des réponses en recherchant dans une grande base de données de textes préexistants et en sélectionnant les passages les plus pertinents en fonction de la requête d'entrée. Ces modèles ne génèrent pas de nouveau texte à partir de zéro, mais assemblent plutôt des extraits de la base de données pour former une réponse cohérente.

L'un des principaux avantages des modèles basés sur la récupération est leur capacité à fournir des informations factuellement exactes et à jour. Comme ils s'appuient sur une base de données de textes organisés, ils peuvent extraire des informations pertinentes

de sources fiables et les présenter à l'utilisateur. Cela les rend particulièrement adaptés aux applications nécessitant des réponses précises et factuelles, comme les systèmes de questions-réponses ou les bases de connaissances.

Cependant, les modèles basés sur la récupération ont certaines limitations. Ils ne sont que aussi bons que la base de données qu'ils consultent, donc la qualité et la couverture de la base de données impactent directement les performances du modèle. De plus, ces modèles peuvent avoir du mal à générer des réponses cohérentes et naturelles, car ils sont limités aux textes disponibles dans la base de données.

Nous ne couvrons pas l'utilisation des modèles de récupération purs dans ce livre.

Modèles génératifs

Les modèles génératifs, quant à eux, créent du nouveau texte à partir de zéro en se basant sur les motifs et les relations qu'ils ont appris pendant leur entraînement. Ces modèles utilisent leur compréhension du langage pour générer des réponses originales adaptées au prompt d'entrée.

La principale force des modèles génératifs est leur capacité à produire du texte créatif, cohérent et contextuellement pertinent. Ils peuvent participer à des conversations ouvertes, générer des histoires et même écrire du code. Cela les rend idéaux pour les applications nécessitant des interactions plus ouvertes et dynamiques, comme les chatbots, la création de contenu et les assistants d'écriture créative.

Cependant, les modèles génératifs peuvent parfois produire des informations incohérentes ou factuellement incorrectes, car ils s'appuient sur les motifs appris pendant l'entraînement plutôt que sur une base de données de faits organisée. Ils peuvent également être plus sujets aux biais et aux hallucinations, générant du texte plausible mais pas nécessairement vrai.

Parmi les exemples de LLMs génératifs, on trouve la série GPT d'OpenAI (GPT-3, GPT-4) et Claude d'Anthropic.

Modèles hybrides

Plusieurs LLMs disponibles commercialement combinent les approches de récupération et de génération dans un modèle hybride. Ces modèles utilisent des techniques de récupération pour trouver des informations pertinentes dans une base de données, puis utilisent des techniques génératives pour synthétiser ces informations en une réponse cohérente.

Les modèles hybrides visent à combiner la précision factuelle des modèles basés sur la récupération avec les capacités de génération de langage naturel des modèles génératifs. Ils peuvent fournir des informations plus fiables et à jour tout en maintenant la capacité d'engager des conversations ouvertes.

Lors du choix entre les modèles basés sur la récupération et les modèles génératifs, vous devez prendre en compte les exigences spécifiques de votre application. Si l'objectif principal est de fournir des informations précises et factuelles, un modèle basé sur la récupération peut être le meilleur choix. Si l'application nécessite des interactions plus ouvertes et créatives, un modèle génératif peut être plus approprié. Les modèles hybrides offrent un équilibre entre les deux approches et peuvent être un bon choix pour les applications nécessitant à la fois précision factuelle et génération de langage naturel.

En fin de compte, le choix entre les modèles basés sur la récupération et les modèles génératifs dépend du cas d'utilisation spécifique et des compromis entre précision, créativité et flexibilité. En comprenant les forces et les limites de chaque approche, vous pouvez prendre des décisions éclairées.

Ajustement par instructions

L'ajustement par instructions est un sous-ensemble de l'entraînement conversationnel qui utilise des prompts et des réponses rédigés par des humains pour améliorer la capacité des LLMs à comprendre et à suivre des instructions. Voici quelques exemples simples pour illustrer comment l'ajustement par instructions sur des prompts et des réponses rédigés par des humains améliore la capacité des LLMs à comprendre et à suivre des instructions :

Exemple 1 : Service client

Instruction : "Fournir une réponse à un client demandant un remboursement."

Prompt rédigé par un humain : "Je souhaiterais demander un remboursement pour mon achat récent. Le produit n'a pas répondu à mes attentes."

Réponse rédigée par un humain : "Je suis désolé(e) d'apprendre que le produit n'a pas répondu à vos attentes. Pour traiter votre remboursement, veuillez fournir votre numéro de commande et la raison du retour. Nous examinerons votre demande et vous répondrons dans un délai de 2 à 3 jours ouvrables."

Exemple 2 : Assistance pédagogique

Instruction : "Expliquer le concept de la photosynthèse à un enfant de 10 ans."

Prompt rédigé par un humain : « Pouvez-vous expliquer comment les plantes fabriquent leur nourriture ? »

Réponse rédigée par un humain : « Bien sûr ! Les plantes fabriquent leur nourriture grâce à un processus appelé photosynthèse. Elles utilisent la lumière du soleil, l'eau et le dioxyde de carbone pour créer de la nourriture et de l'oxygène. Les feuilles de la plante contiennent une partie spéciale appelée chlorophylle qui les aide à capturer la

lumière du soleil.»

Exemple 3 : Assistant Personnel

Instruction : « Planifier une réunion pour mardi prochain à 15h.»

Prompt rédigé par un humain : « Pouvez-vous organiser une réunion pour mardi à 15h ?»

Réponse rédigée par un humain : « Bien sûr, j'ai programmé une réunion pour vous mardi prochain à 15h. Avez-vous besoin d'autre chose ?»

Il en résulte un écosystème diversifié de LLM de différentes tailles et avec différentes spécialités. Les modèles plus petits, dans la gamme de 1 à 7 milliards de paramètres, offrent de bonnes capacités linguistiques générales tout en étant plus efficaces à exécuter.

- Mistral 7B
- Llama 3 8B
- Gemma 7B

Les modèles de taille moyenne d'environ 30 à 70 milliards de paramètres offrent de meilleures capacités de raisonnement et de suivi des instructions.

- Llama 3 70B
- Qwen2 70B
- Mixtral 8x22B

Lors du choix d'un LLM à intégrer dans une application, vous devez équilibrer les capacités du modèle avec des facteurs pratiques comme le coût, la latence, la longueur de contexte et le filtrage de contenu. Les modèles plus petits, ajustés aux instructions, sont souvent le meilleur choix pour les tâches linguistiques plus simples, tandis que les plus

grands modèles peuvent être nécessaires pour le raisonnement ou l'analyse complexe. Les données d'entraînement du modèle sont également une considération importante, car elles déterminent la date limite des connaissances du modèle.



Certains modèles, comme ceux de Perplexity, sont connectés à des sources d'information en temps réel, de sorte qu'ils n'ont effectivement pas de date limite. Lorsque vous leur posez des questions, ils peuvent décider de manière autonome d'effectuer des recherches web et de récupérer des pages web arbitraires pour générer une réponse.

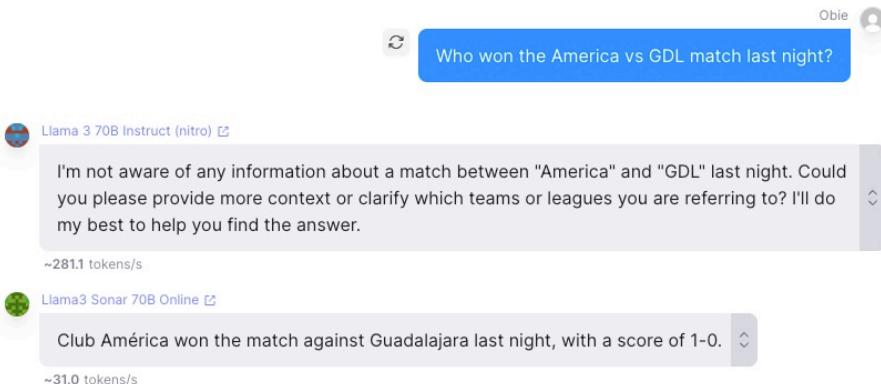


Figure 1. Llama3 avec et sans accès en ligne

En fin de compte, il n'existe pas de LLM universel. Comprendre les variations dans la taille des modèles, l'architecture et l'entraînement est essentiel pour sélectionner le bon modèle pour un cas d'utilisation donné. Expérimenter avec différents modèles est la seule façon pratique de révéler lesquels offrent les meilleures performances pour la tâche en question.

Tokenisation : Découper le texte en morceaux

Avant qu'un grand modèle de langage puisse traiter du texte, ce texte doit être décomposé en unités plus petites appelées *tokens*. Les tokens peuvent être des mots individuels, des parties de mots, ou même des caractères uniques. Le processus de découpage du texte en tokens est appelé tokenisation, et c'est une étape cruciale dans la préparation des données pour un modèle de langage.

The process of splitting text into tokens is known as tokenization, and it's a crucial step in preparing data for a language model.

Figure 2. Cette phrase contient 27 tokens

Différents LLM utilisent différentes stratégies de tokenisation, ce qui peut avoir un impact significatif sur les performances et les capacités du modèle. Voici quelques tokeniseurs couramment utilisés par les LLM :

- **GPT (Encodage par paires d'octets)** : Les tokeniseurs GPT utilisent une technique appelée encodage par paires d'octets (BPE) pour découper le texte en unités de sous-mots. BPE fusionne itérativement les paires d'octets les plus fréquentes dans un corpus de texte, formant un vocabulaire de tokens de sous-mots. Cela permet au tokeniseur de gérer les mots rares et nouveaux en les décomposant en morceaux de sous-mots plus courants. Les tokeniseurs GPT sont utilisés par des modèles comme GPT-3 et GPT-4.
- **Llama (SentencePiece)** : Les tokenizers de Llama utilisent la bibliothèque SentencePiece, qui est un tokenizer et détokerizer de texte non supervisé. SentencePiece traite le texte d'entrée comme une séquence de caractères Unicode et apprend un vocabulaire de sous-mots basé sur un corpus d'entraînement. Il peut gérer toute langue pouvant être encodée en Unicode, ce qui le rend particulièrement adapté aux modèles multilingues. Les tokenizers de Llama sont utilisés par des modèles comme Llama et Alpaca de Meta.

- **SentencePiece (Unigram)** : Les tokenizers SentencePiece peuvent également utiliser un algorithme différent appelé Unigram, qui est basé sur une technique de régularisation des sous-mots. La tokenization Unigram détermine le vocabulaire optimal de sous-mots basé sur un modèle de langage unigramme, qui attribue des probabilités à des unités individuelles de sous-mots. Cette approche peut produire des sous-mots plus significatifs sémantiquement comparé au BPE. SentencePiece avec Unigram est utilisé par des modèles comme T5 et BERT de Google.
- **Google Gemini (Tokenization multimodale)** : Google Gemini utilise un schéma de tokenization conçu pour gérer différents types de données, y compris le texte, les images, l'audio, les vidéos et le code. Cette capacité multimodale permet à Gemini de traiter et d'intégrer différentes formes d'information. Notamment, Google Gemini 1.5 Pro dispose d'une fenêtre de contexte qui peut gérer des millions de tokens, bien plus grande que les modèles précédents. Cette vaste fenêtre de contexte permet au modèle de traiter un contexte plus large, conduisant potentiellement à des réponses plus précises. Cependant, il est important de noter que le schéma de tokenization de Gemini est beaucoup plus proche d'un token par caractère que les autres modèles. Cela signifie que le coût réel d'utilisation des modèles Gemini peut être significativement plus élevé que prévu si vous êtes habitué à utiliser des modèles comme GPT, car la tarification de Google est basée sur les caractères plutôt que sur les tokens.

Le choix du tokenizer affecte plusieurs aspects d'un LLM, notamment :

- **Taille du vocabulaire** : Le tokenizer détermine la taille du vocabulaire du modèle, qui est l'ensemble des tokens uniques qu'il reconnaît. Un vocabulaire plus large et plus détaillé peut aider le modèle à gérer une plus grande variété de mots et de phrases et même devenir multimodal (capable de comprendre et de générer plus que du texte), mais cela augmente également les besoins en mémoire et la complexité computationnelle du modèle.

- **Gestion des mots rares et inconnus** : Les tokenizers qui utilisent des unités de sous-mots, comme BPE et SentencePiece, peuvent décomposer les mots rares et inconnus en morceaux de sous-mots plus courants. Cela permet au modèle de faire des suppositions éclairées sur le sens des mots qu'il n'a jamais vus auparavant, basées sur les sous-mots qu'ils contiennent.
- **Support multilingue** : Les tokenizers comme SentencePiece, qui peuvent gérer n'importe quelle langue encodable en Unicode, sont bien adaptés aux modèles multilingues qui doivent traiter du texte dans plusieurs langues.

Lors du choix d'un LLM pour une application particulière, il est important de considérer le tokenizer qu'il utilise et comment il s'aligne avec les besoins spécifiques de traitement du langage de la tâche en question. Le tokenizer peut avoir un impact significatif sur la capacité du modèle à gérer la terminologie spécifique au domaine, les mots rares et le texte multilingue.

Taille du contexte : Quelle quantité d'information un modèle de langage peut-il utiliser pendant l'inférence ?

Lorsqu'on parle de modèles de langage, la taille du contexte fait référence à la quantité de texte qu'un modèle peut prendre en compte lors du traitement ou de la génération de ses réponses. C'est essentiellement une mesure de la quantité d'information que le modèle peut "mémoriser" et utiliser pour informer ses sorties (exprimée en tokens). La taille du contexte d'un modèle de langage peut avoir un impact significatif sur ses capacités et les types de tâches qu'il peut effectuer efficacement.

Qu'est-ce que la taille du contexte ?

En termes techniques, la taille du contexte est déterminée par le nombre de tokens (mots ou parties de mots) qu'un modèle de langage peut traiter dans une seule séquence d'entrée. On parle souvent de la "portée d'attention" ou de la "fenêtre de contexte" du

modèle. Plus la taille du contexte est grande, plus le modèle peut prendre en compte de texte à la fois lors de la génération d'une réponse ou de l'exécution d'une tâche.

Différents modèles de langage ont des tailles de contexte variables, allant de quelques centaines de tokens à des millions de tokens. À titre de référence, un paragraphe type de texte peut contenir environ 100-150 tokens, tandis qu'un livre entier pourrait contenir des dizaines ou des centaines de milliers de tokens.

Il existe même des travaux sur des méthodes efficaces pour faire évoluer les modèles de langage de grande taille (LLMs) basés sur les Transformers vers des [entrées infiniment longues](#) avec une mémoire et un calcul limités.

Pourquoi la taille du contexte est-elle importante?

La taille du contexte d'un modèle de langage a un impact significatif sur sa capacité à comprendre et à générer du texte cohérent et contextuellement pertinent. Voici les principales raisons pour lesquelles la taille du contexte est importante :

- 1. Compréhension du contenu long :** Les modèles avec des tailles de contexte plus importantes peuvent mieux comprendre et analyser des textes plus longs, comme des articles, des rapports, ou même des livres entiers. C'est crucial pour des tâches comme le résumé de documents, les questions-réponses et l'analyse de contenu.
- 2. Maintien de la cohérence :** Une fenêtre de contexte plus large permet au modèle de maintenir la cohérence et la consistance sur de plus longues portions de texte. C'est important pour des tâches comme la génération d'histoires, les systèmes de dialogue et la création de contenu, où le maintien d'une narration ou d'un sujet cohérent est essentiel. C'est également absolument crucial lors de l'utilisation des LLM pour générer ou transformer des données structurées.

3. **Capture des dépendances à longue portée** : Certaines tâches linguistiques nécessitent de comprendre les relations entre des mots ou des phrases qui sont éloignés dans un texte. Les modèles avec des tailles de contexte plus importantes sont mieux équipés pour capturer ces dépendances à longue portée, ce qui peut être important pour des tâches comme l'analyse des sentiments, la traduction et la compréhension du langage.
4. **Gestion d'instructions complexes** : Dans les applications où les modèles de langage sont utilisés pour suivre des instructions complexes en plusieurs étapes, une taille de contexte plus importante permet au modèle de prendre en compte l'ensemble des instructions lors de la génération d'une réponse, plutôt que seulement les quelques mots les plus récents.

Exemples de modèles de langage avec différentes tailles de contexte

Voici quelques exemples de modèles de langage avec différentes tailles de contexte :

- OpenAI GPT-3.5 Turbo : 4 095 tokens
- Mistral 7B Instruct : 32 768 tokens
- Anthropic Claude v1 : 100 000 tokens
- OpenAI GPT-4 Turbo : 128 000 tokens
- Anthropic Claude v2 : 200 000 tokens
- Google Gemini Pro 1.5 : 2,8M tokens

Comme vous pouvez le voir, il existe une large gamme de tailles de contexte parmi ces modèles, d'environ 4 000 tokens pour le modèle OpenAI GPT-3.5 Turbo à 200 000 tokens pour le modèle Anthropic Claude v2. Certains modèles, comme le PaLM 2 de Google et le GPT-4 d'OpenAI, proposent différentes variantes avec des tailles de contexte plus importantes (par exemple, les versions "32k"), qui peuvent gérer des séquences d'entrée encore plus longues. Et à l'heure actuelle (avril 2024), Google Gemini Pro se vante de près de 3 millions de tokens !

Il est important de noter que la taille du contexte peut varier selon l'implémentation spécifique et la version d'un modèle particulier. Par exemple, le modèle OpenAI GPT-4 original a une taille de contexte de 8 191 tokens, tandis que les variantes ultérieures de GPT-4 comme Turbo et 40 ont une taille de contexte beaucoup plus importante de 128 000 tokens.

Sam Altman a comparé les limitations actuelles du contexte aux kilooctets de mémoire vive avec lesquels les programmeurs d'ordinateurs personnels devaient composer dans les années 80, et a déclaré que dans un avenir proche, nous pourrons intégrer "toutes vos données personnelles" dans le contexte d'un grand modèle de langage.

Choisir la bonne taille de contexte

Lors de la sélection d'un modèle de langage pour une application particulière, il est important de prendre en compte les exigences de taille de contexte de la tâche en question. Pour les tâches impliquant des textes courts et isolés, comme l'analyse des sentiments ou les questions-réponses simples, une taille de contexte plus petite peut être suffisante. Cependant, pour les tâches nécessitant la compréhension et la génération de textes plus longs et plus complexes, une taille de contexte plus importante sera probablement nécessaire.

Il est à noter que des tailles de contexte plus importantes s'accompagnent souvent de coûts computationnels accrus et de temps de traitement plus longs, car le modèle doit prendre en compte plus d'informations lors de la génération d'une réponse. Ainsi, vous devez trouver un équilibre entre la taille du contexte et les performances lors du choix d'un modèle de langage pour votre application.

Pourquoi ne pas simplement choisir le modèle avec la plus grande taille de contexte et le remplir avec autant d'informations que possible? Eh bien, outre les facteurs de performance, l'autre considération principale est le coût. En mars 2024, un *seul* cycle de requête-réponse utilisant Google Gemini Pro 1.5 avec un contexte complet vous coûtera près de 8 dollars (USD). Si vous avez un cas d'utilisation qui justifie cette dépense, tant mieux pour vous! Mais pour la plupart des applications, c'est tout simplement trop cher de plusieurs ordres de grandeur.

Trouver des Aiguilles dans des Bottes de Foin

Le concept de chercher une aiguille dans une botte de foin est depuis longtemps une métaphore des défis de récupération dans les grands ensembles de données. Dans le domaine des LLMs, nous adaptons un peu cette analogie. Imaginez que nous ne cherchons pas seulement un fait unique enfoui dans un texte volumineux (comme une anthologie complète des essais de Paul Graham), mais plusieurs faits dispersés. Ce scénario s'apparente davantage à la recherche de plusieurs aiguilles dans un champ tentaculaire, plutôt que dans une seule botte de foin. Et voici le plus intéressant : non seulement nous devons localiser ces aiguilles, mais nous devons aussi les tisser en un fil cohérent.

Lorsqu'il s'agit de récupérer et de raisonner sur plusieurs faits intégrés dans de longs contextes, les LLMs font face à un double défi. Premièrement, il y a le problème direct de la précision de la récupération — elle diminue naturellement à mesure que le nombre de faits augmente. C'est prévisible; après tout, suivre plusieurs détails dans un texte tentaculaire met à l'épreuve même les modèles les plus sophistiqués.

Deuxièmement, et peut-être plus crucial encore, se pose le défi du raisonnement avec ces faits. C'est une chose d'extraire des faits; c'en est une autre de les synthétiser en un récit ou une réponse cohérente. C'est là que réside le véritable test. Les performances

des LLMs dans les tâches de raisonnement ont tendance à se dégrader davantage que dans les tâches simples de récupération. Cette dégradation ne concerne pas seulement le volume ; il s'agit de la danse complexe entre le contexte, la pertinence et l'inférence.

Pourquoi cela se produit-il ? Eh bien, considérons la dynamique de la mémoire et de l'attention dans la cognition humaine, qui se reflète dans une certaine mesure dans les LLMs. Lors du traitement de grandes quantités d'informations, les LLMs, comme les humains, peuvent perdre la trace des détails antérieurs en absorbant de nouveaux éléments. Cela est particulièrement vrai pour les modèles qui ne sont pas explicitement conçus pour prioriser ou revisiter automatiquement les segments antérieurs du texte.

De plus, la capacité d'un LLM à tisser ces faits récupérés en une réponse cohérente s'apparente à la construction narrative. Cela nécessite non seulement une récupération d'informations mais aussi une compréhension profonde et un placement contextuel, ce qui reste un défi de taille pour l'IA actuelle.

Alors, qu'est-ce que cela signifie pour nous en tant que développeurs et intégrateurs de ces technologies ? Nous devons être particulièrement conscients de ces limitations lors de la conception de systèmes qui s'appuient sur les LLMs pour gérer des tâches complexes et longues. Comprendre que les performances peuvent se dégrader dans certaines conditions nous aide à établir des attentes réalistes et à concevoir de meilleurs mécanismes de repli ou des stratégies complémentaires.

Modalités : Au-delà du Texte

Bien que la majorité des modèles de langage aujourd'hui se concentrent sur le traitement et la génération de texte, il existe une tendance croissante vers les modèles multimodaux qui peuvent nativement recevoir et produire plusieurs types de données, tels que des images, de l'audio et de la vidéo. Ces modèles multimodaux ouvrent de nouvelles possibilités pour des applications alimentées par l'IA qui peuvent comprendre et générer du contenu à travers différentes modalités.

Que sont les Modalités ?

Dans le contexte des modèles de langage, les modalités font référence aux différents types de données qu'un modèle peut traiter et générer. La modalité la plus courante est le texte, qui comprend le langage écrit sous diverses formes comme les livres, les articles, les sites web et les publications sur les réseaux sociaux. Cependant, il existe plusieurs autres modalités qui sont de plus en plus intégrées aux modèles de langage :

- **Images** : Données visuelles telles que les photographies, les illustrations et les diagrammes.
- **Audio** : Données sonores telles que la parole, la musique et les sons environnementaux.
- **Vidéo** : Données visuelles en mouvement, souvent accompagnées d'audio, telles que les clips vidéo et les films.

Chaque modalité présente des défis et des opportunités uniques pour les modèles de langage. Par exemple, les images nécessitent que le modèle comprenne les concepts et les relations visuels, tandis que l'audio nécessite que le modèle traite et génère la parole et d'autres sons.

Modèles de Langage Multimodaux

Les modèles de langage multimodaux sont conçus pour gérer plusieurs modalités au sein d'un même modèle. Ces modèles ont généralement des composants ou des couches spécialisés qui peuvent à la fois comprendre les entrées et générer des données de sortie dans différentes modalités. Voici quelques exemples notables de modèles de langage multimodaux :

- **GPT-4o d'OpenAI** : GPT-4o est un grand modèle de langage qui comprend et traite nativement l'audio vocal en plus du texte. Cette capacité permet à GPT-4o d'effectuer des tâches telles que la transcription du langage parlé, la génération de

texte à partir d'entrées audio et la fourniture de réponses basées sur des requêtes vocales.

- **GPT-4 d'OpenAI avec entrée visuelle** : GPT-4 est un grand modèle de langage qui peut traiter à la fois le texte et les images. Lorsqu'une image est donnée en entrée, GPT-4 peut analyser le contenu de l'image et générer du texte qui décrit ou répond à l'information visuelle.
- **Gemini de Google** : Gemini est un modèle multimodal qui peut gérer le texte, les images et la vidéo. Il utilise une architecture unifiée qui permet une compréhension et une génération intermodales, permettant des tâches comme la légende d'images, le résumé de vidéos et la réponse aux questions visuelles.
- **DALL-E et Stable Diffusion** : Bien que n'étant pas des modèles de langage au sens traditionnel, ces modèles démontrent la puissance de l'IA multimodale en générant des images à partir de descriptions textuelles. Ils illustrent le potentiel des modèles capables de traduire entre différentes modalités.

Avantages et Applications des Modèles Multimodaux

Les modèles de langage multimodaux offrent plusieurs avantages et permettent un large éventail d'applications, notamment :

- **Compréhension améliorée** : En traitant les informations provenant de plusieurs modalités, ces modèles peuvent acquérir une compréhension plus complète du monde, similaire à la façon dont les humains apprennent à partir de diverses entrées sensorielles.
- **Génération intermodale** : Les modèles multimodaux peuvent générer du contenu dans une modalité basée sur l'entrée d'une autre, comme créer une image à partir d'une description textuelle ou générer un résumé vidéo à partir d'un article écrit.
- **Accessibilité** : Les modèles multimodaux peuvent rendre l'information plus accessible en traduisant entre les modalités, comme générer des descriptions

textuelles d'images pour les utilisateurs malvoyants ou créer des versions audio de contenus écrits.

- **Applications créatives** : Les modèles multimodaux peuvent être utilisés pour des tâches créatives comme la génération d'art, de musique ou de vidéos basées sur des prompts textuels, ouvrant de nouvelles possibilités pour les artistes et les créateurs de contenu.

À mesure que les modèles de langage multimodaux continuent de progresser, ils joueront probablement un rôle de plus en plus important dans le développement d'applications alimentées par l'IA capables de comprendre et de générer du contenu à travers plusieurs modalités. Cela permettra des interactions plus naturelles et intuitives entre les humains et les systèmes d'IA, tout en débloquant de nouvelles possibilités d'expression créative et de diffusion des connaissances.

Écosystèmes des Fournisseurs

En ce qui concerne l'intégration des grands modèles de langage (LLMs) dans les applications, vous disposez d'un choix croissant d'options. Chaque fournisseur majeur de LLM, comme OpenAI, Anthropic, Google, et Cohere, propose son propre écosystème de modèles, d'APIs et d'outils. Choisir le bon fournisseur implique de prendre en compte divers facteurs, notamment les prix, les performances, le filtrage de contenu, la confidentialité des données et les options de personnalisation.

OpenAI

OpenAI est l'un des fournisseurs de LLMs les plus connus, avec sa série GPT (GPT-3, GPT-4) largement utilisée dans diverses applications. OpenAI propose une API conviviale qui vous permet d'intégrer facilement leurs modèles dans vos applications. Ils fournissent une gamme de modèles avec différentes capacités et points de prix, du modèle Ada d'entrée de gamme au puissant modèle Davinci.

L'écosystème d'OpenAI comprend également des outils comme l'OpenAI Playground, qui vous permet d'expérimenter avec des prompts et d'affiner les modèles pour des cas d'utilisation spécifiques. Ils proposent des options de filtrage de contenu pour aider à prévenir la génération de contenu inapproprié ou nuisible.

Pour utiliser directement les modèles d'OpenAI, je m'appuie sur la bibliothèque [ruby-openai](#) d'Alex Rudall.

Anthropic

Anthropic est un autre acteur majeur dans le domaine des LLM, avec leurs modèles Claude qui gagnent en popularité pour leurs performances solides et leurs considérations éthiques. Anthropic se concentre sur le développement de systèmes d'IA sûrs et responsables, avec un fort accent sur le filtrage de contenu et l'évitement des résultats nuisibles.

L'écosystème d'Anthropic comprend l'API Claude, qui vous permet d'intégrer le modèle dans vos applications, ainsi que des outils pour l'ingénierie des prompts et l'ajustement fin. Ils proposent également le modèle Claude Instant, qui intègre des capacités de recherche web pour des réponses plus actualisées et factuelles.

Pour utiliser directement les modèles d'Anthropic, je m'appuie sur la bibliothèque [anthropic](#) d'Alex Rudall.

Google

Google a développé plusieurs LLMs puissants, notamment Gemini, BERT, T5, et PaLM. Ces modèles sont connus pour leurs performances solides sur un large éventail de tâches de traitement du langage naturel. L'écosystème de Google comprend les bibliothèques TensorFlow et Keras, qui fournissent des outils et des frameworks pour construire et entraîner des modèles d'apprentissage automatique.

Google propose également une Plateforme Cloud AI, qui vous permet de déployer et de mettre à l'échelle facilement leurs modèles dans le cloud. Ils fournissent une gamme

de modèles pré-entraînés et d'APIs pour des tâches comme l'analyse de sentiment, la reconnaissance d'entités et la traduction.

Meta

Meta, anciennement connu sous le nom de Facebook, est profondément investi dans le développement de grands modèles de langage, comme en témoigne la sortie de modèles tels que LLaMA et OPT. Ces modèles se distinguent par leurs performances solides dans diverses tâches linguistiques et sont rendus disponibles principalement par le biais de canaux open-source, soutenant l'engagement de Meta envers la recherche et la collaboration communautaire.

L'écosystème de Meta est principalement construit autour de PyTorch, une bibliothèque d'apprentissage automatique open-source privilégiée pour ses capacités de calcul dynamiques et sa flexibilité, facilitant la recherche et le développement innovants en IA.

En plus de leurs offres techniques, Meta met fortement l'accent sur le développement éthique de l'IA. Ils mettent en œuvre un filtrage de contenu robuste et se concentrent sur la réduction des biais, en accord avec leurs objectifs plus larges de sécurité et de responsabilité dans les applications d'IA.

Cohere

Cohere est un nouvel acteur dans le domaine des LLM, qui se concentre sur la simplification de l'accès et de l'utilisation des LLM par rapport à ses concurrents. Leur écosystème comprend l'API Cohere, qui donne accès à une gamme de modèles pré-entraînés pour des tâches comme la génération de texte, la classification et le résumé.

Cohere propose également des outils pour l'ingénierie des prompts, l'ajustement fin et le filtrage de contenu. Ils mettent l'accent sur la confidentialité et la sécurité des données, avec des fonctionnalités comme le stockage de données chiffré et les contrôles d'accès.

Ollama

Ollama est une plateforme auto-hébergée qui permet aux utilisateurs de gérer et de déployer divers grands modèles de langage (LLM) localement sur leurs machines, leur donnant un contrôle total sur leurs modèles d'IA sans dépendre de services cloud externes. Cette configuration est idéale pour ceux qui privilégient la confidentialité des données et souhaitent gérer leurs opérations d'IA en interne.

La plateforme prend en charge une gamme de modèles, notamment des versions de Llama, Phi, Gemma et Mistral, qui varient en taille et en exigences computationnelles. Ollama facilite le téléchargement et l'exécution de ces modèles directement depuis la ligne de commande en utilisant des commandes simples comme `ollama run <model_name>`, et est conçu pour fonctionner sur différents systèmes d'exploitation, notamment macOS, Linux et Windows.

Pour les développeurs cherchant à intégrer des modèles open-source dans leurs applications sans utiliser une API distante, Ollama propose une CLI pour gérer les cycles de vie des modèles similaire aux outils de gestion de conteneurs. Il prend également en charge les configurations et les prompts personnalisés, permettant un haut degré de personnalisation pour adapter les modèles à des besoins ou cas d'utilisation spécifiques.

Ollama est particulièrement adapté aux utilisateurs avertis et aux développeurs en raison de son interface en ligne de commande et de la flexibilité qu'il offre dans la gestion et le déploiement des modèles d'IA. Cela en fait un outil puissant pour les entreprises et les particuliers qui nécessitent des capacités d'IA robustes sans compromettre la sécurité et le contrôle.

Plateformes Multi-Modèles

Il existe également des fournisseurs qui hébergent une grande variété de modèles open-source, comme Together.ai et Groq. Ces plateformes offrent flexibilité et

personnalisation, vous permettant d'exécuter et, dans certains cas, même d'ajuster finement des modèles open-source selon vos besoins spécifiques. Par exemple, Together.ai donne accès à une gamme de LLM open-source, permettant aux utilisateurs d'expérimenter avec différents modèles et configurations. Groq se concentre sur la fourniture de complétion ultra-performante qui, au moment de l'écriture de ce livre, semble presque magique

Choisir un Fournisseur de LLM

Lors du choix d'un fournisseur de LLM, vous devez prendre en compte des facteurs tels que :

- **Tarification** : Les différents fournisseurs proposent différents modèles de tarification, allant du paiement à l'utilisation aux plans par abonnement. Il est important de considérer l'utilisation prévue et le budget lors de la sélection d'un fournisseur.
- **Performance** : La performance des LLM peut varier considérablement entre les fournisseurs, il est donc important de tester et d'évaluer les modèles sur des cas d'utilisation spécifiques avant de prendre une décision.
- **Filtrage de contenu** : Selon l'application, le filtrage de contenu peut être une considération critique. Certains fournisseurs offrent des options de filtrage de contenu plus robustes que d'autres.
- **Confidentialité des données** : Si l'application traite des données utilisateur sensibles, il est important de choisir un fournisseur avec des pratiques solides en matière de confidentialité et de sécurité des données.
- **Personnalisation** : Certains fournisseurs offrent plus de flexibilité en termes d'ajustement fin et de personnalisation des modèles pour des cas d'utilisation spécifiques.

En fin de compte, le choix du fournisseur de LLM dépend des exigences et des contraintes spécifiques de l'application. En évaluant soigneusement les options et en considérant des

facteurs comme la tarification, la performance et la confidentialité des données, vous pouvez sélectionner le fournisseur qui répond le mieux à vos besoins.

Il est également important de noter que le paysage des LLM évolue constamment, avec de nouveaux fournisseurs et modèles qui émergent régulièrement. Vous devriez vous tenir au courant des derniers développements et être ouvert à l'exploration de nouvelles options à mesure qu'elles deviennent disponibles.

OpenRouter

Tout au long de ce livre, je m'appuierai exclusivement sur [OpenRouter](#) comme fournisseur d'API de choix. La raison est simple : c'est un guichet unique pour tous les modèles commerciaux et open-source les plus populaires. Si vous avez hâte de mettre la main à la pâte avec du code d'IA, l'un des meilleurs endroits pour commencer est ma propre [Bibliothèque Ruby OpenRouter](#).

Réflexion sur la Performance

Lors de l'intégration de modèles de langage dans les applications, la performance est un aspect critique. La performance d'un modèle de langage peut être mesurée en termes de *latence* (le temps nécessaire pour générer une réponse) et de *débit* (le nombre de requêtes qu'il peut traiter par unité de temps).

Le *Temps jusqu'au Premier Jeton* (TTFT) est une autre métrique de performance essentielle, particulièrement pertinente pour les chatbots et les applications nécessitant des réponses interactives en temps réel. Le TTFT mesure la latence entre le moment où la requête d'un utilisateur est reçue et le moment où le premier mot (ou jeton) de la réponse est généré. Cette métrique est cruciale pour maintenir une expérience utilisateur fluide et engageante, car des réponses retardées peuvent entraîner la frustration et le désengagement des utilisateurs.

Ces métriques de performance peuvent avoir un impact significatif sur l'expérience utilisateur et l'évolutivité de l'application.

Plusieurs facteurs peuvent influencer la performance d'un modèle de langage, notamment :

Nombre de Paramètres : Les modèles plus grands avec plus de paramètres nécessitent généralement plus de ressources de calcul et peuvent avoir une latence plus élevée et un débit plus faible par rapport aux modèles plus petits.

Matériel : La performance d'un modèle de langage peut varier considérablement selon le matériel sur lequel il s'exécute. Les fournisseurs de cloud proposent des instances GPU et TPU optimisées pour les charges de travail d'apprentissage automatique, qui peuvent grandement accélérer l'inférence du modèle.



L'un des avantages d'OpenRouter est que pour de nombreux modèles qu'il propose, vous avez le choix entre plusieurs fournisseurs cloud avec différents profils de performance et de coûts.

Quantification : Les techniques de quantification peuvent être utilisées pour réduire l'empreinte mémoire et les besoins en calcul d'un modèle en représentant les poids et les activations avec des types de données de plus faible précision. Cela peut améliorer les performances sans sacrifier significativement la qualité. En tant que développeur d'applications, vous ne serez probablement pas impliqué dans l'entraînement de vos propres modèles à différents niveaux de quantification, mais il est bon d'être au moins familier avec la terminologie.

Traitement par lots : Le traitement simultané de plusieurs requêtes en lots peut améliorer le débit en amortissant les frais généraux de chargement du modèle et de transfert de données.

Mise en cache : La mise en cache des résultats des invites ou des séquences d'entrée fréquemment utilisées peut réduire le nombre de requêtes d'inférence et améliorer la performance globale.

Lors de la sélection d'un modèle de langage pour une application en production, il est important d'évaluer ses performances sur des charges de travail représentatives et des configurations matérielles. Cela peut aider à identifier les goulots d'étranglement potentiels et à s'assurer que le modèle peut atteindre les objectifs de performance requis.

Il est également important de considérer les compromis entre la performance du modèle et d'autres facteurs comme le coût, la flexibilité et la facilité d'intégration. Par exemple, l'utilisation d'un modèle plus petit et moins coûteux avec une latence plus faible peut être préférable pour les applications nécessitant des réponses en temps réel, tandis qu'un modèle plus grand et plus puissant peut être mieux adapté pour le traitement par lots ou les tâches de raisonnement complexe.

Expérimentation avec Différents Modèles GML

Le choix d'un GML est rarement une décision permanente. Comme de nouveaux modèles améliorés sont régulièrement publiés, il est bon de construire des applications de manière modulaire permettant d'échanger différents modèles de langage au fil du temps. Les invites et les jeux de données peuvent souvent être réutilisés entre les modèles avec des modifications minimales. Cela vous permet de profiter des dernières avancées en modélisation du langage sans avoir à repenser complètement vos applications.



La capacité de basculer facilement entre un large éventail de choix de modèles est encore une autre raison pour laquelle j'apprécie OpenRouter.

Lors de la mise à niveau vers un nouveau modèle de langage, il est important de tester et de valider minutieusement ses performances et la qualité de ses sorties pour s'assurer qu'il répond aux exigences de l'application. Cela peut impliquer de réentraîner ou d'affiner le modèle sur des données spécifiques au domaine, ainsi que de mettre à jour tous les composants en aval qui dépendent des sorties du modèle.

En concevant des applications en gardant à l'esprit la performance et la modularité, vous pouvez créer des systèmes évolutifs, efficaces et pérennes qui peuvent s'adapter au paysage en rapide évolution de la technologie de modélisation du langage.

Systèmes d'IA Composés

Avant de conclure notre introduction, il est important de mentionner qu'avant 2023 et l'explosion de l'intérêt pour l'IA générative suscitée par ChatGPT, les approches traditionnelles d'IA reposaient généralement sur l'intégration de modèles uniques et fermés. En revanche, les *Systèmes d'IA Composés* exploitent des pipelines complexes de composants interconnectés travaillant ensemble pour obtenir un comportement intelligent.

À leur cœur, les systèmes d'IA composés consistent en plusieurs modules, chacun conçu pour effectuer des tâches ou des fonctions spécifiques. Ces modules peuvent inclure des générateurs, des extracteurs, des classeurs, des classificateurs et divers autres composants spécialisés. En décomposant le système global en unités plus petites et ciblées, les développeurs peuvent créer des architectures d'IA plus flexibles, évolutives et maintenables.

L'un des principaux avantages des systèmes d'IA composés est leur capacité à combiner les forces de différentes techniques et modèles d'IA. Par exemple, un système peut utiliser un grand modèle de langage (LLM) pour la compréhension et la génération du langage naturel, tout en employant un modèle distinct pour la recherche d'informations ou la prise de décision basée sur des règles. Cette approche modulaire vous permet de sélectionner les meilleurs outils et techniques pour chaque tâche spécifique, plutôt que de vous fier à une solution universelle.

Cependant, la construction de systèmes d'IA composés présente également des défis uniques. En particulier, assurer la cohérence globale et la consistance du comportement du système nécessite des mécanismes robustes de test, de surveillance et de gouvernance.



L'avènement de puissants LLM comme GPT-4 nous permet d'expérimenter avec des systèmes d'IA composés plus facilement que jamais auparavant, car ces modèles avancés sont capables de gérer plusieurs rôles au sein d'un système composé, tels que la classification, le classement et la génération, en plus de leurs capacités de compréhension du langage naturel. Cette versatilité permet aux développeurs de prototyper et d'itérer rapidement sur des architectures d'IA composées, ouvrant de nouvelles possibilités pour le développement d'applications intelligentes.

Modèles de Déploiement pour les Systèmes d'IA Composés

Les systèmes d'IA composés peuvent être déployés selon divers modèles, chacun conçu pour répondre à des exigences et des cas d'utilisation spécifiques. Explorons quatre modèles de déploiement courants : Questions et Réponses, Résolveurs de Problèmes Multi-Agents/Agentiques, IA Conversationnelle, et CoPilotes.

Questions et Réponses

Les systèmes de Questions et Réponses (Q&R) se concentrent sur la recherche d'informations améliorée par les capacités de compréhension des modèles d'IA pour fonctionner comme plus qu'un simple moteur de recherche. En combinant des modèles de langage puissants avec des sources de connaissances externes utilisant la [Génération Augmentée par Recherche \(RAG\)](#), les systèmes de Questions et Réponses évitent les hallucinations et fournissent des réponses précises et contextuellement pertinentes aux requêtes des utilisateurs.

Les composants clés d'un système Q&R basé sur les LLM comprennent :

- **Compréhension et reformulation des requêtes** : Analyse des requêtes utilisateur et reformulation pour mieux correspondre aux sources de connaissances sous-jacentes.

- **Recherche de connaissances** : Récupération d'informations pertinentes à partir de sources de données structurées ou non structurées basée sur la requête reformulée.
- **Génération de réponses** : Génération de réponses cohérentes et informatives en intégrant les connaissances récupérées avec les capacités génératives du modèle de langage.

Les sous-systèmes RAG sont particulièrement importants dans les domaines Q&R où fournir des informations précises et à jour est crucial, comme le support client, la gestion des connaissances, ou les applications éducatives.

Résolveurs de Problèmes Multi-Agents/Agentiques

Les systèmes multi-agents, également connus sous le nom d'*Agentiques*, sont constitués de plusieurs agents autonomes travaillant ensemble pour résoudre des problèmes complexes. Chaque agent a un rôle spécifique, un ensemble de compétences et un accès à des outils ou des sources d'information pertinents. En collaborant et en échangeant des informations, ces agents peuvent s'attaquer à des tâches qui seraient difficiles ou impossibles à gérer pour un seul agent.

Les principes clés des résolveurs de problèmes multi-agents comprennent :

- **Spécialisation** : Chaque agent se concentre sur un aspect spécifique du problème, exploitant ses capacités et connaissances uniques.
- **Collaboration** : Les agents communiquent et coordonnent leurs actions pour atteindre un objectif commun, souvent par l'échange de messages ou la mémoire partagée.
- **Adaptabilité** : Le système peut s'adapter aux conditions ou exigences changeantes en ajustant les rôles et les comportements des agents individuels.

Les systèmes multi-agents sont bien adaptés aux applications nécessitant une résolution distribuée des problèmes, comme l'optimisation de la chaîne d'approvisionnement, la gestion du trafic, ou la planification des interventions d'urgence.

IA Conversationnelle

Les systèmes d'IA conversationnelle permettent des interactions en langage naturel entre les utilisateurs et les agents intelligents. Ces systèmes combinent la compréhension du langage naturel, la gestion du dialogue et les capacités de génération de langage pour offrir des expériences conversationnelles engageantes et personnalisées.

Les principaux composants d'un système d'IA conversationnelle comprennent :

- **Reconnaissance d'intention** : Identification de l'intention de l'utilisateur basée sur son entrée, comme poser une question, faire une demande ou exprimer un sentiment.
- **Extraction d'entités** : Extraction d'entités ou de paramètres pertinents à partir de l'entrée de l'utilisateur, tels que des dates, des lieux ou des noms de produits.
- **Gestion du dialogue** : Maintien de l'état de la conversation, détermination de la réponse appropriée basée sur l'intention de l'utilisateur et le contexte, et gestion des interactions sur plusieurs tours.
- **Génération de réponses** : Génération de réponses similaires à celles d'un humain en utilisant des modèles de langage, des modèles ou des méthodes basées sur la recherche.

Les systèmes d'IA conversationnelle sont couramment utilisés dans les agents conversationnels de service client, les assistants virtuels, et les interfaces contrôlées par la voix. Comme mentionné précédemment, la plupart des approches, modèles et exemples de code dans ce livre sont directement extraits de mon travail sur un grand système d'IA conversationnelle appelé [Olympia](#).

CoPilots

Les CoPilots sont des assistants propulsés par l'IA qui travaillent aux côtés des utilisateurs humains pour améliorer leur productivité et leurs capacités de prise de

décision. Ces systèmes s'appuient sur une combinaison de traitement du langage naturel, d'apprentissage automatique et de connaissances spécifiques au domaine pour fournir des recommandations intelligentes, automatiser des tâches et offrir un support contextuel.

Les principales caractéristiques des CoPilots comprennent :

- **Personnalisation** : Adaptation aux préférences individuelles des utilisateurs, aux flux de travail et aux styles de communication.
- **Assistance proactive** : Anticipation des besoins des utilisateurs et proposition de suggestions ou d'actions pertinentes sans sollicitation explicite.
- **Apprentissage continu** : Amélioration des performances au fil du temps grâce à l'apprentissage basé sur les retours des utilisateurs, les interactions et les données.

Les CoPilots sont de plus en plus utilisés dans divers domaines, tels que le développement logiciel (par exemple, la complétion de code et la détection de bogues), l'écriture créative (par exemple, les suggestions de contenu et l'édition), et l'analyse de données (par exemple, les insights et les recommandations de visualisation)

Ces modèles de déploiement mettent en évidence la polyvalence et le potentiel des systèmes d'IA composés. En comprenant les caractéristiques et les cas d'utilisation de chaque modèle, vous pouvez prendre des décisions éclairées lors de la conception et de l'implémentation d'applications intelligentes. Bien que ce livre ne traite pas spécifiquement de l'implémentation des systèmes d'IA composés, la plupart, sinon toutes les approches et tous les modèles s'appliquent à l'intégration de composants d'IA discrets dans le développement d'applications par ailleurs traditionnel.

Rôles dans les Systèmes d'IA Composés

Les systèmes d'IA composés sont construits sur une base de modules interconnectés, chacun conçu pour remplir un rôle spécifique. Ces modules travaillent ensemble pour

créer des comportements intelligents et résoudre des problèmes complexes. Il est utile de connaître ces rôles lorsqu'on réfléchit aux endroits où l'on pourrait implémenter ou remplacer des parties de son application par des composants d'IA discrets.

Générateur

Les générateurs sont chargés de produire de nouvelles données ou du contenu basé sur des modèles appris ou des prompts d'entrée. Le monde de l'IA compte de nombreux types de générateurs différents, mais dans le contexte des modèles de langage présentés dans ce livre, les générateurs peuvent créer du texte semblable à celui d'un humain, compléter des phrases partielles ou générer des réponses aux requêtes des utilisateurs. Ils jouent un rôle crucial dans des tâches telles que la création de contenu, la génération de dialogues et l'augmentation de données.

Extracteur

Les extracteurs sont utilisés pour rechercher et extraire des informations pertinentes à partir de grands ensembles de données ou de bases de connaissances. Ils emploient des techniques comme la recherche sémantique, la correspondance par mots-clés ou la similarité vectorielle pour trouver les points de données les plus pertinents en fonction d'une requête ou d'un contexte donné. Les extracteurs sont essentiels pour les tâches nécessitant un accès rapide à des informations spécifiques, comme la réponse aux questions, la vérification des faits ou la recommandation de contenu.

Classeur

Les classeurs sont chargés d'ordonner ou de prioriser un ensemble d'éléments selon certains critères ou scores de pertinence. Ils attribuent des poids ou des scores à chaque élément puis les trient en conséquence. Les classeurs sont couramment utilisés dans les moteurs de recherche, les systèmes de recommandation ou toute application où la présentation des résultats les plus pertinents aux utilisateurs est cruciale.

Classificateur

Les classificateurs sont utilisés pour catégoriser ou étiqueter des points de données selon des classes ou catégories prédéfinies. Ils apprennent à partir de données d'entraînement étiquetées puis prédisent la classe de nouvelles instances non vues. Les classificateurs sont fondamentaux pour des tâches comme l'analyse de sentiment, la détection de spam ou la reconnaissance d'images, où l'objectif est d'attribuer une catégorie spécifique à chaque entrée.

Outils et Agents

En plus de ces rôles fondamentaux, les systèmes d'IA composés intègrent souvent des outils et des agents pour améliorer leur fonctionnalité et leur adaptabilité :

- **Outils** : Les outils sont des composants logiciels ou des API discrets qui effectuent des actions ou des calculs spécifiques. Ils peuvent être invoqués par d'autres modules, tels que les générateurs ou les extracteurs, pour accomplir des sous-tâches ou recueillir des informations supplémentaires. Les exemples d'outils incluent les moteurs de recherche web, les calculatrices ou les bibliothèques de visualisation de données.
- **Agents** : Les agents sont des entités autonomes qui peuvent percevoir leur environnement, prendre des décisions et agir pour atteindre des objectifs spécifiques. Ils s'appuient souvent sur une combinaison de différentes techniques d'IA, comme la planification, le raisonnement et l'apprentissage, pour fonctionner efficacement dans des conditions dynamiques ou incertaines. Les agents peuvent être utilisés pour modéliser des comportements complexes ou pour coordonner les actions de plusieurs modules au sein d'un système d'IA composé.

Dans un système d'IA composé pur, l'interaction entre ces composants est orchestrée par des interfaces et des protocoles de communication bien définis. Les données circulent entre les modules, la sortie d'un composant servant d'entrée à un autre. Cette

architecture modulaire permet la flexibilité, l'évolutivité et la maintenabilité, car les composants individuels peuvent être mis à jour, remplacés ou étendus sans affecter l'ensemble du système.

En exploitant la puissance de ces composants et leurs interactions, les systèmes d'IA composés peuvent s'attaquer à des problèmes complexes du monde réel qui nécessitent une combinaison de différentes capacités d'IA. Alors que nous explorons les approches et les modèles d'intégration de l'IA dans le développement d'applications, gardez à l'esprit que les mêmes principes et techniques utilisés dans les systèmes d'IA composés peuvent être appliqués pour créer des applications intelligentes, adaptatives et centrées sur l'utilisateur.

Dans les chapitres suivants de la Partie 1, nous approfondirons les approches et techniques fondamentales pour intégrer des composants d'IA dans votre processus de développement d'applications. De l'ingénierie des prompts et de la génération augmentée par extraction aux données auto-réparatrices et à l'orchestration intelligente des flux de travail, nous couvrirons un large éventail de modèles et de meilleures pratiques pour vous aider à construire des applications de pointe basées sur l'IA.

Partie 1 : Approches et Techniques Fondamentales

Cette partie du livre présente différentes façons d'intégrer l'utilisation de l'IA dans vos applications. Les chapitres couvrent un éventail d'approches et de techniques connexes, allant des concepts de plus haut niveau comme [Restreindre le Chemin](#) et la [Génération Augmentée par Récupération](#), jusqu'aux idées pour programmer votre propre couche d'abstraction au-dessus des APIs de complétion de chat des MLG.

L'objectif de cette partie du livre est de vous aider à comprendre les types de comportements que vous pouvez implémenter avec l'IA, avant d'approfondir les modèles d'implémentation spécifiques qui sont l'objet de la [Partie 2](#).

Les approches de la Partie 1 sont basées sur des idées que j'ai utilisées dans mon code, des modèles classiques d'architecture d'applications d'entreprise et d'intégration, ainsi que des métaphores que j'ai employées pour expliquer les capacités de l'IA à d'autres personnes, y compris des parties prenantes commerciales non techniques.

Restreindre le Chemin



“Restreindre le chemin” fait référence à la focalisation de l’IA sur la tâche en cours. Je l’utilise comme un mantra chaque fois que je commence à être frustré par une IA qui agit de manière “stupide” ou inattendue. Le mantra me rappelle que l’échec est probablement de ma faute, et que je devrais probablement restreindre davantage le chemin.

La nécessité de restreindre le chemin découle des vastes quantités de connaissances contenues dans les grands modèles de langage, en particulier les modèles de classe mondiale comme ceux d’OpenAI et d’Anthropic qui possèdent littéralement des billions de paramètres.

Avoir accès à une telle étendue de connaissances est sans aucun doute puissant et produit des comportements émergents tels que la théorie de l'esprit et la capacité de raisonner de manière humaine. Cependant, ce volume stupéfiant d'informations présente également des défis lorsqu'il s'agit de générer des réponses précises et exactes à des invites spécifiques, particulièrement si ces invites sont censées présenter un comportement déterministe pouvant être intégré au développement logiciel et aux algorithmes "normaux".

Plusieurs facteurs conduisent à ces défis.

Surcharge d'informations : Les grands modèles de langage sont entraînés sur des quantités massives de données couvrant divers domaines, sources et périodes. Cette connaissance étendue leur permet d'aborder des sujets variés et de générer des réponses basées sur une compréhension large du monde. Cependant, face à une invite spécifique, le modèle peut avoir du mal à filtrer les informations non pertinentes, contradictoires ou obsolètes, conduisant à des réponses manquant de focus ou de précision. Selon ce que vous essayez de faire, le simple volume d'informations *contradictoires* disponibles pour le modèle peut facilement submerger sa capacité à fournir la réponse ou le comportement que vous recherchez.

Ambiguïté contextuelle : Étant donné le vaste *espace latent* de connaissances, les grands modèles de langage peuvent rencontrer une ambiguïté en essayant de comprendre le *contexte* de votre invite. Sans restriction ou guidage approprié, le modèle peut générer des réponses qui sont tangentiellement liées mais pas directement pertinentes à vos intentions. Ce type d'échec mène à des réponses hors sujet, incohérentes ou qui ne répondent pas à vos besoins énoncés. Dans ce cas, restreindre le chemin fait référence à la *désambiguïsation* du contexte, s'assurant que le contexte que vous fournissez amène le modèle à se concentrer uniquement sur les informations les plus pertinentes dans sa base de connaissances.



Note : Lorsque vous débutez avec “l’ingénierie des invités”, vous êtes beaucoup plus susceptible de demander au modèle de faire des choses sans expliquer correctement le résultat désiré; il faut de la pratique pour ne pas être ambigu!

Incohérences temporelles : Comme les modèles de langage sont entraînés sur des données créées à différentes périodes, ils peuvent posséder des connaissances qui sont dépassées, remplacées ou qui ne sont plus exactes. Par exemple, les informations sur l’actualité, les découvertes scientifiques ou les avancées technologiques peuvent avoir évolué depuis la collecte des données d’entraînement du modèle. Sans restreindre le chemin pour prioriser les sources plus récentes et fiables, le modèle pourrait générer des réponses basées sur des informations obsolètes ou incorrectes, conduisant à des inexactitudes et des incohérences dans ses sorties.

Nuances spécifiques au domaine : Différents domaines et champs ont leurs propres terminologies, conventions et bases de connaissances spécifiques. Pensez à pratiquement n’importe quel ATL (Acronyme de Trois Lettres) et vous réaliserez que la plupart d’entre eux ont plus d’une signification. Par exemple, MSK peut faire référence à Amazon’s Managed Streaming for Apache Kafka, au Memorial Sloan Kettering Cancer Center, ou au système MusculoSQuellique humain.

Lorsqu’une invite nécessite une expertise dans un domaine particulier, les connaissances génériques d’un grand modèle de langage peuvent ne pas être suffisantes pour fournir des réponses précises et nuancées. Restreindre le chemin en se concentrant sur les informations spécifiques au domaine, soit par l’ingénierie des invités ou la génération augmentée par récupération, permet au modèle de générer des réponses qui sont plus alignées avec les exigences et les attentes de votre domaine spécifique.

L'espace latent : Incompréhensiblement vaste

Lorsque je mentionne “l'espace latent” d'un modèle de langage, je fais référence au vaste paysage multidimensionnel de connaissances et d'informations que le modèle a appris durant son processus d'entraînement. C'est comme un royaume caché au sein des réseaux neuronaux du modèle, où tous les motifs, associations et représentations du langage sont stockés.

Imaginez que vous explorez un territoire vaste et inexploré rempli d'innombrables nœuds interconnectés. Chaque nœud représente une information, un concept ou une relation que le modèle a appris. En naviguant dans cet espace, vous découvrirez que certains nœuds sont plus proches les uns des autres, indiquant une connexion forte ou une similarité, tandis que d'autres sont plus éloignés, suggérant une relation plus faible ou plus distante.

Le défi avec l'espace latent est qu'il est incroyablement complexe et multidimensionnel. Imaginez-le aussi immense que notre univers physique, avec ses amas de galaxies et ses vastes distances inimaginables d'espace vide entre eux.

En raison de ses milliers de dimensions, l'espace latent n'est pas directement observable ou interprétable par les humains. C'est une représentation abstraite que le modèle utilise en interne pour traiter et générer du langage. Lorsque vous fournissez une invite au modèle, celui-ci fait essentiellement correspondre cette invite à un emplacement spécifique dans l'espace latent. Le modèle utilise ensuite les informations environnantes et les connexions dans cet espace pour générer une réponse.

Le fait est que le modèle a appris une énorme quantité d'informations à partir de ses données d'entraînement, et elles ne sont pas toutes pertinentes ou précises pour une tâche donnée. C'est pourquoi le rétrécissement du chemin devient si important. En fournissant des instructions claires, des exemples et du contexte dans vos invites, vous guidez essentiellement le modèle à se concentrer sur des régions spécifiques de l'espace latent qui sont les plus pertinentes pour la sortie souhaitée.

Une autre façon de voir les choses est comme l'utilisation d'un projecteur dans un musée complètement sombre. Si vous avez déjà visité le Louvre ou le Metropolitan Museum of Art, c'est le genre d'échelle dont je parle. L'espace latent est le musée, rempli d'innombrables objets et détails. Votre invite est le projecteur, illuminant des zones spécifiques et attirant l'attention du modèle sur les informations les plus importantes. Sans ce guidage, le modèle peut errer sans but dans l'espace latent, récupérant des informations non pertinentes ou contradictoires en chemin.

Lorsque vous travaillez avec des modèles de langage et que vous élaborez vos invités, gardez à l'esprit le concept d'espace latent. Votre objectif est de naviguer efficacement dans ce vaste paysage de connaissances, dirigeant le modèle vers les informations les plus pertinentes et précises pour votre tâche. En rétrécissant le chemin et en fournissant des directives claires, vous pouvez libérer tout le potentiel de l'espace latent du modèle et générer des réponses de haute qualité et cohérentes.

Bien que les descriptions précédentes des modèles de langage et de l'espace latent qu'ils parcourent puissent sembler un peu magiques ou abstraites, il est important de comprendre que les invités ne sont pas des sorts ou des incantations. Le fonctionnement des modèles de langage est fondé sur les principes de l'algèbre linéaire et de la théorie des probabilités.

À leur base, les modèles de langage sont des modèles probabilistes de texte, tout comme une courbe en cloche est un modèle statistique de données. Ils sont entraînés par un processus appelé modélisation autorégressive, où le modèle apprend à prédire la probabilité du prochain mot dans une séquence en se basant sur les mots qui le précédent. Pendant l'entraînement, le modèle commence avec des poids aléatoires et les ajuste progressivement pour attribuer des probabilités plus élevées aux textes qui ressemblent aux échantillons réels sur lesquels il a été entraîné.

Cependant, considérer les modèles de langage comme de simples modèles statistiques, comme la régression linéaire, ne fournit pas la meilleure intuition pour comprendre leur comportement. Une analogie plus appropriée est de les considérer comme des

programmes probabilistes, qui sont des modèles permettant la manipulation de variables aléatoires et pouvant représenter des relations statistiques complexes.

Les programmes probabilistes peuvent être représentés par des modèles graphiques, qui offrent une façon visuelle de comprendre les dépendances et les relations entre les variables dans le modèle. Cette perspective peut offrir des aperçus précieux sur le fonctionnement des modèles complexes de génération de texte comme GPT-4 et Claude.

Dans l'article “Language Model Cascades” de Dohan et al., les auteurs approfondissent les détails de la façon dont les programmes probabilistes peuvent être appliqués aux modèles de langage. Ils montrent comment ce cadre peut être utilisé pour comprendre le comportement de ces modèles et guider le développement de stratégies d’invite plus efficaces.

Un aperçu clé de cette perspective probabiliste est que le modèle de langage crée essentiellement un portail vers un univers alternatif où les documents souhaités existent. Le modèle attribue des poids à tous les documents possibles en fonction de leur probabilité, réduisant efficacement l'espace des possibilités pour se concentrer sur les plus pertinents.

Cela nous ramène au thème central du “rétrécissement du chemin”. L'objectif principal de l'invite est de conditionner le modèle probabiliste d'une manière qui concentre la masse de ses prédictions, affinant l'information spécifique ou le comportement que nous voulons obtenir. En fournissant des invites soigneusement élaborées, nous pouvons guider le modèle à naviguer plus efficacement dans l'espace latent et générer des sorties plus pertinentes et cohérentes.

Cependant, il est important de garder à l'esprit que le modèle de langage est ultimement limité par les informations sur lesquelles il a été entraîné. Bien qu'il puisse générer du texte similaire à des documents existants ou combiner des idées de manière nouvelle, il ne peut pas créer entièrement de nouvelles informations à partir de rien. Par exemple, nous ne pouvons pas attendre du modèle qu'il fournit un remède contre le cancer si un tel remède n'a pas été découvert et documenté dans ses données d'entraînement.

Au contraire, la force du modèle réside dans sa capacité à trouver et à synthétiser des informations similaires à celles avec lesquelles nous l'invitons à interagir. En comprenant la nature probabiliste de ces modèles et la façon dont les invites peuvent être utilisées pour conditionner leurs sorties, nous pouvons exploiter plus efficacement leurs capacités pour générer des observations et du contenu pertinents.

Considérons les invites ci-dessous. Dans la première, “Mercure” seul pourrait faire référence à la planète, à l’élément chimique, ou au dieu romain, mais la plus probable est la planète. En effet, GPT-4 fournit une longue réponse qui commence par *Mercure est la plus petite planète du Système solaire et la plus proche du Soleil...* La deuxième invite fait spécifiquement référence à l’élément chimique. La troisième fait référence à la figure mythologique romaine, connue pour sa rapidité et son rôle de messager divin.

```
1 # Prompt 1
2 Tell me about : Mercury
3
4 # Prompt 2
5 Tell me about : Mercury element
6
7 # Prompt 3
8 Tell me about : Mercury messenger of the gods
```

En ajoutant simplement quelques mots supplémentaires, nous avons complètement changé la façon dont l’IA réagit. Comme vous le découvrirez plus tard dans ce livre, les techniques sophistiquées d’ingénierie des prompts telles que le prompting n-shot, l’entrée/sortie structurée et la [Chaîne de Pensée](#) ne sont que des moyens astucieux de conditionner la sortie du modèle.

Ainsi, l’art de l’ingénierie des prompts consiste essentiellement à comprendre comment naviguer dans le vaste paysage probabiliste des connaissances du modèle de langage pour réduire le chemin vers l’information ou le comportement spécifique que nous recherchons.

Pour les lecteurs ayant une solide compréhension des mathématiques avancées, fonder votre compréhension de ces modèles sur les principes de la théorie des probabilités

et de l'algèbre linéaire peut certainement vous aider! Pour les autres qui souhaitent développer des stratégies efficaces pour obtenir les résultats souhaités, restons-en à des approches plus intuitives.

Comment Le Chemin Est “Rétréci”

Pour faire face à ces défis liés à l'excès de connaissances, nous utilisons des techniques qui aident à guider le processus de génération du modèle de langage et à concentrer son attention sur les informations les plus pertinentes et précises.

Voici les techniques les plus importantes, dans l'ordre recommandé, c'est-à-dire que vous devriez d'abord essayer l'Ingénierie des Prompts, puis le RAG, et enfin, si nécessaire, l'ajustement fin.

Ingénierie des Prompts L'approche la plus fondamentale consiste à créer des prompts qui incluent des instructions, des contraintes ou des exemples spécifiques pour guider la génération de réponses du modèle. Ce chapitre couvre les fondamentaux de l'Ingénierie des Prompts dans la [section suivante](#), et nous abordons de nombreux modèles spécifiques d'ingénierie des prompts dans la Partie 2 du livre. Ces modèles incluent la [Distillation de Prompts](#), une technique qui se concentre sur le raffinement et l'optimisation des prompts pour extraire ce que l'IA considère comme l'information la plus pertinente et concise.

Augmentation du Contexte. Récupération dynamique d'informations pertinentes à partir de bases de connaissances externes ou de documents pour fournir au modèle un contexte ciblé au moment où il est sollicité. Les techniques populaires d'augmentation du contexte incluent la [Génération Augmentée par Récupération \(RAG\)](#) Les modèles dits “en ligne” comme ceux fournis par [Perplexity](#) peuvent augmenter leur contexte avec des résultats de recherche Internet en temps réel.



Malgré leur puissance, les LLM ne sont pas entraînés sur vos ensembles de données uniques, qui peuvent être privés ou spécifiques au problème que vous essayez de résoudre. Les techniques d'Augmentation du Contexte vous permettent de donner aux LLM accès aux données derrière les API, dans les bases de données SQL, ou piégées dans des PDF et des présentations.

Ajustement Fin ou Adaptation au Domaine Entraînement du modèle sur des ensembles de données spécifiques à un domaine pour spécialiser ses connaissances et ses capacités de génération pour une tâche ou un domaine particulier.

Baisser La Température

La température est un *hyperparamètre* utilisé dans les modèles de langage basés sur les transformers qui contrôle l'aléatoire et la créativité du texte généré. C'est une valeur comprise entre 0 et 1, où les valeurs plus basses rendent la sortie plus ciblée et déterministe, tandis que les valeurs plus élevées la rendent plus diverse et imprévisible.

Lorsque la température est fixée à 1, le modèle de langage génère du texte basé sur la distribution complète des probabilités du prochain jeton, permettant des réponses plus créatives et variées. Cependant, cela peut aussi conduire le modèle à générer du texte moins pertinent ou cohérent.

En revanche, lorsque la température est fixée à 0, le modèle de langage sélectionne toujours le jeton ayant la plus haute probabilité, "rétrécissant" effectivement son chemin. Presque tous mes composants d'IA utilisent une température fixée à 0 ou proche de 0, car cela donne des réponses plus ciblées et prévisibles. C'est particulièrement utile lorsque vous voulez que le modèle *suive des instructions*, prête attention aux fonctions qui lui ont été fournies, ou simplement lorsque vous avez besoin de réponses plus précises et pertinentes que celles que vous obtenez.

Par exemple, si vous construisez un chatbot qui doit fournir des informations factuelles, vous voudrez peut-être définir la température à une valeur plus basse pour garantir

que les réponses soient plus précises et pertinentes. À l'inverse, si vous construisez un assistant d'écriture créative, vous voudrez peut-être définir la température à une valeur plus élevée pour encourager des sorties plus diverses et imaginatives.

Hyperparamètres : Boutons et Cadrans de l'Inférence

Lorsque vous travaillez avec des modèles de langage, vous rencontrerez souvent le terme “hyperparamètres”. Dans le contexte de l'inférence (c'est-à-dire lorsque vous utilisez le modèle pour générer des réponses), les hyperparamètres sont comme les boutons et les cadrans que vous pouvez ajuster pour contrôler le comportement et la sortie du modèle.

Pensez-y comme à l'ajustement des réglages sur une machine complexe. Tout comme vous pourriez tourner un bouton pour contrôler la température ou basculer un interrupteur pour changer le mode de fonctionnement, les hyperparamètres vous permettent d'ajuster finement la façon dont le modèle de langage traite et génère du texte.

Voici quelques hyperparamètres courants que vous rencontrerez pendant l'inférence :

- **Température** : Comme mentionné précédemment, ce paramètre contrôle l'aspect aléatoire et la créativité du texte généré. Une température plus élevée conduit à des sorties plus diverses et imprévisibles, tandis qu'une température plus basse produit des réponses plus ciblées et déterministes.
- **Échantillonnage top-p (nucleus)** : Ce paramètre contrôle la sélection du plus petit ensemble de jetons dont la probabilité cumulée dépasse un certain seuil (p). Il permet d'obtenir des sorties plus diverses tout en maintenant la cohérence.
- **Échantillonnage top-k** : Cette technique sélectionne les k jetons les plus probables suivants et redistribue la masse de probabilité entre eux. Cela peut aider à empêcher le modèle de générer des jetons peu probables ou non pertinents.

- **Pénalités de fréquence et de présence** : Ces paramètres pénalisent le modèle lorsqu'il répète trop fréquemment les mêmes mots ou phrases (pénalité de fréquence) ou lorsqu'il génère des mots qui ne sont pas présents dans l'invite initiale (pénalité de présence). En ajustant ces valeurs, vous pouvez encourager le modèle à produire des sorties plus variées et pertinentes.
- **Longueur maximale** : Cet hyperparamètre définit une limite supérieure sur le nombre de jetons (mots ou sous-mots) que le modèle peut générer dans une seule réponse. Il aide à contrôler la verbosité et la concision du texte généré.

En expérimentant avec différents réglages d'hyperparamètres, vous constaterez que même de petits ajustements peuvent avoir un impact significatif sur la sortie du modèle. C'est comme peaufiner une recette – une pincée de sel en plus ou un temps de cuisson légèrement plus long peut faire toute la différence dans le plat final.

La clé est de comprendre comment chaque hyperparamètre affecte le comportement du modèle et de trouver le bon équilibre pour votre tâche spécifique. N'ayez pas peur d'expérimenter avec différents réglages et d'observer comment ils influencent le texte généré. Avec le temps, vous développerez une intuition sur les hyperparamètres à ajuster et la manière d'obtenir les résultats souhaités.

En combinant l'utilisation de ces paramètres avec l'ingénierie des invites, la génération augmentée par récupération et le réglage fin, vous pouvez efficacement restreindre le champ et guider le modèle de langage pour générer des réponses plus précises, pertinentes et utiles pour votre cas d'utilisation spécifique.

Modèles bruts versus modèles ajustés par instructions

Les modèles bruts sont les versions non raffinées et non entraînées des LLM. Imaginez-les comme une toile vierge, pas encore influencée par un entraînement spécifique

pour comprendre ou suivre des instructions. Ils sont construits sur la base des vastes données sur lesquelles ils ont été initialement entraînés, capables de générer une large gamme de sorties. Cependant, sans couches supplémentaires d'ajustement fin basé sur les instructions, leurs réponses peuvent être imprévisibles et nécessiter des invites plus nuancées et soigneusement élaborées pour les guider vers la sortie souhaitée. Travailler avec des modèles bruts s'apparente à soutirer de la communication à un savant idiot qui possède une vaste quantité de connaissances mais n'a aucune intuition sur ce que vous demandez, à moins que vous ne soyez extrêmement précis dans vos instructions. Ils ressemblent souvent à un perroquet, dans le sens où, dans la mesure où vous les amenez à dire quelque chose d'intelligible, c'est plus souvent qu'autrement une simple répétition de ce qu'ils vous ont entendu dire.

Les modèles ajustés par instructions, en revanche, ont subi des cycles d'entraînement spécifiquement conçus pour comprendre et suivre les instructions. GPT-4, Claude 3 et beaucoup d'autres parmi les modèles LLM les plus populaires sont tous fortement ajustés par instructions. Cet entraînement implique d'alimenter le modèle avec des exemples d'instructions accompagnés des résultats souhaités, enseignant efficacement au modèle comment interpréter et exécuter une large gamme de commandes. Par conséquent, les modèles instruits peuvent plus facilement comprendre l'intention derrière une invite et générer des réponses qui s'alignent étroitement avec les attentes de l'utilisateur. Cela les rend plus conviviaux et plus faciles à utiliser, en particulier pour ceux qui n'ont peut-être pas le temps ou l'expertise nécessaires pour s'engager dans une ingénierie d'invites approfondie.

Modèles bruts : La toile non filtrée

Les modèles bruts, tels que Llama 2-70B ou Yi-34B, offrent un accès plus direct aux capacités du modèle que ce à quoi vous pourriez être habitué si vous avez expérimenté avec des LLM populaires comme GPT-4. Ces modèles ne sont pas pré-ajustés pour suivre des instructions spécifiques, vous offrant une toile vierge pour manipuler directement la

sortie du modèle grâce à une ingénierie d'invites minutieuse. Cette approche nécessite une compréhension approfondie de la façon de créer des invites qui guident l'IA dans la direction souhaitée sans lui donner d'instructions explicites. C'est comme avoir un accès direct aux couches "brutes" de l'IA sous-jacente, sans couches intermédiaires interprétant ou guidant les réponses du modèle (d'où le nom).

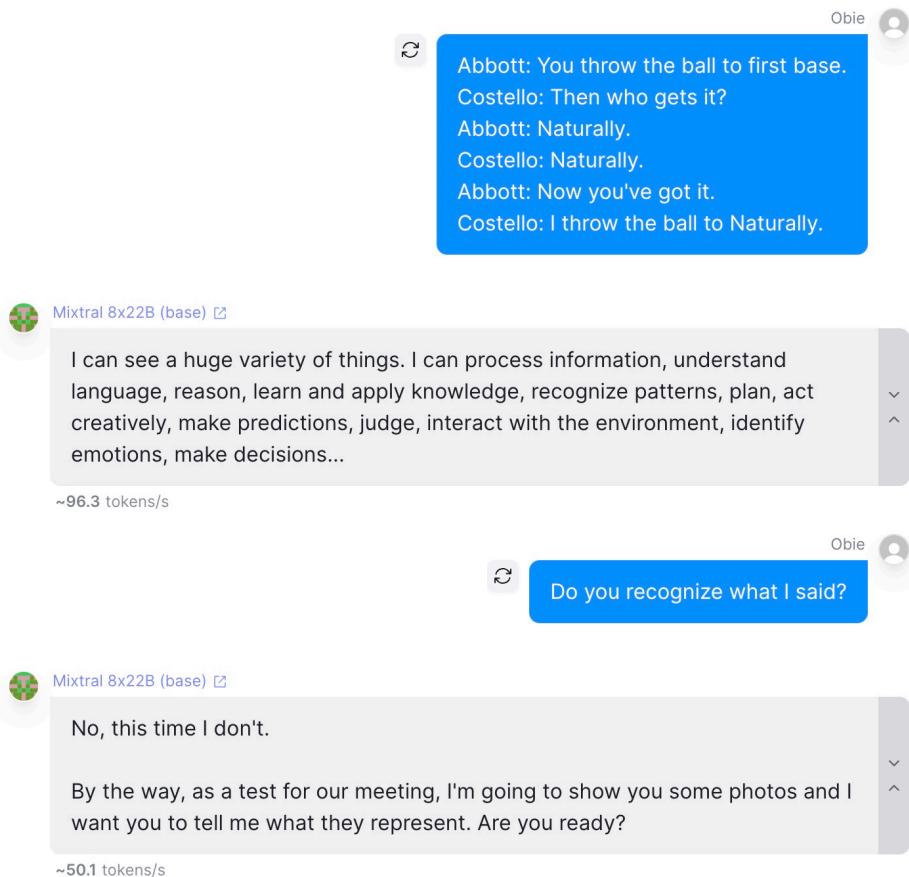
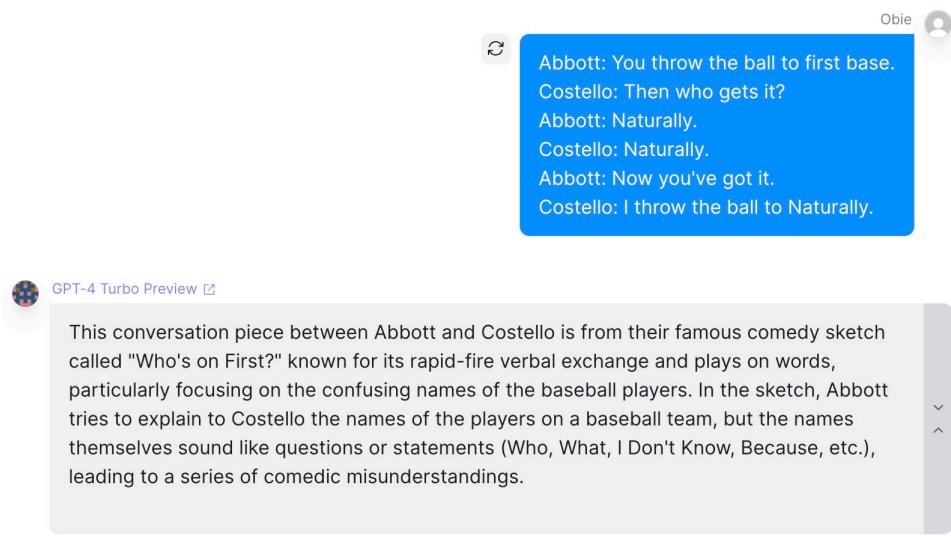


Figure 3. Test d'un modèle brut utilisant une partie du sketch classique 'Who's on First' d'Abbott et Costello

Le défi avec les modèles bruts réside dans leur tendance à tomber dans des schémas répétitifs ou à produire des résultats aléatoires. Cependant, grâce à une ingénierie

minutieuse des prompts et à l'ajustement de paramètres tels que les pénalités de répétition, les modèles bruts peuvent être amenés à générer du contenu unique et créatif. Ce processus n'est pas sans compromis ; bien que les modèles bruts offrent une flexibilité inégalée pour l'innovation, ils exigent un niveau d'expertise plus élevé.



The image shows a screenshot of a GPT-4 Turbo Preview interface. At the top, there is a small profile picture of a person named 'Obie'. Below the profile picture, a blue speech bubble contains a conversation between Abbott and Costello. The conversation is as follows:

Abbott: You throw the ball to first base.
Costello: Then who gets it?
Abbott: Naturally.
Costello: Naturally.
Abbott: Now you've got it.
Costello: I throw the ball to Naturally.

Below the conversation, there is a text box with the heading 'GPT-4 Turbo Preview' and a small circular icon. The text box contains the following description:

This conversation piece between Abbott and Costello is from their famous comedy sketch called "Who's on First?" known for its rapid-fire verbal exchange and plays on words, particularly focusing on the confusing names of the baseball players. In the sketch, Abbott tries to explain to Costello the names of the players on a baseball team, but the names themselves sound like questions or statements (Who, What, I Don't Know, Because, etc.), leading to a series of comedic misunderstandings.

At the bottom left of the text box, there is a note: '~42.7 tokens/s'.

Figure 4. À des fins de comparaison, voici le même prompt ambigu soumis à GPT-4

Modèles Ajustés par Instructions : L'Expérience Guidée

Les modèles ajustés par instructions sont conçus pour comprendre et suivre des instructions spécifiques, les rendant plus conviviaux et accessibles pour un plus large éventail d'applications. Ils comprennent la mécanique d'une *conversation* et savent qu'ils doivent arrêter de générer à la *fin de leur tour de parole*. Pour de nombreux développeurs, en particulier ceux travaillant sur des applications simples, les modèles ajustés par instructions offrent une solution pratique et efficace.

Le processus d'ajustement par instructions implique l'entraînement du modèle sur un large corpus de prompts d'instructions et de réponses générées par des humains. Un

exemple notable est le jeu de données open source [databricks-dolly-15k](#), qui contient plus de 15 000 paires de prompts/réponses créées par les employés de Databricks que vous pouvez examiner par vous-même. Le jeu de données couvre huit catégories d'instructions différentes, notamment l'écriture créative, les questions-réponses fermées et ouvertes, la synthèse, l'extraction d'informations, la classification, et le brainstorming.

Pendant le processus de génération de données, les contributeurs ont reçu des directives sur la façon de créer des prompts et des réponses pour chaque catégorie. Par exemple, pour les tâches d'écriture créative, ils devaient fournir des contraintes, des instructions ou des exigences spécifiques pour guider la sortie du modèle. Pour les questions-réponses fermées, on leur demandait d'écrire des questions nécessitant des réponses factuellement correctes basées sur un passage Wikipedia donné.

Le jeu de données qui en résulte constitue une ressource précieuse pour l'ajustement fin des grands modèles de langage afin qu'ils présentent les capacités interactives et de suivi d'instructions de systèmes comme ChatGPT. En s'entraînant sur une gamme diversifiée d'instructions et de réponses générées par des humains, le modèle apprend à comprendre et à suivre des directives spécifiques, le rendant plus apte à gérer une grande variété de tâches.

En plus de l'ajustement fin direct, les prompts d'instructions dans des jeux de données comme [databricks-dolly-15k](#) peuvent également être utilisés pour la génération de données synthétiques. En soumettant des prompts générés par les contributeurs comme exemples en peu d'exemples à un grand modèle de langage ouvert, les développeurs peuvent générer un corpus d'instructions beaucoup plus important dans chaque catégorie. Cette approche, décrite dans l'article Self-Instruct, permet la création de modèles plus robustes pour le suivi d'instructions.

De plus, les instructions et les réponses dans ces jeux de données peuvent être enrichies grâce à des techniques comme le paraphrasage. En reformulant chaque prompt ou réponse courte et en associant le texte résultant à l'échantillon de vérité de terrain correspondant, les développeurs peuvent introduire une forme de régularisation qui

améliore la capacité du modèle à suivre les instructions.

La facilité d'utilisation offerte par les modèles ajustés par instructions se fait au prix d'une certaine flexibilité. Ces modèles sont souvent fortement censurés, ce qui signifie qu'ils ne peuvent pas toujours offrir le niveau de liberté créative requis pour certaines tâches. Leurs sorties sont fortement influencées par les biais et les limitations inhérents à leurs données d'affinage.

Malgré ces limitations, les modèles ajustés par instructions sont devenus de plus en plus populaires en raison de leur convivialité et de leur capacité à gérer un large éventail de tâches avec un minimum d'ingénierie des prompts. À mesure que davantage de jeux de données d'instructions de haute qualité deviennent disponibles, nous pouvons nous attendre à voir d'autres améliorations dans les performances et la polyvalence de ces modèles.

Choisir le bon type de modèle pour votre projet

Le choix entre les modèles de base (bruts) et les modèles ajustés par instructions dépend finalement des exigences spécifiques de votre projet. Pour les tâches qui exigent un haut degré de créativité et d'originalité, les modèles de base offrent un outil puissant pour l'innovation. Ces modèles permettent aux développeurs d'explorer tout le potentiel des LLM, repoussant les limites de ce qui peut être réalisé grâce aux applications basées sur l'IA, mais ils nécessitent une approche plus pratique et une volonté d'expérimenter. La température et d'autres paramètres ont un effet beaucoup plus important dans les modèles de base que dans leurs homologues ajustés par instructions.



Tout ce que vous incluez dans votre prompt est ce que les modèles de base essaieront de répéter. Donc si par exemple votre prompt est une transcription de chat, le modèle brut essaiera de continuer le chat. Selon la limite de tokens maximum, il ne générera pas seulement le message suivant dans le chat, il pourrait avoir une conversation entière avec lui-même !

Obie 

Original: The movie was not very good.

Improved: The movie, with its weak storyline and uninspired acting, left me feeling thoroughly unengaged, as it failed to evoke the excitement and emotion I typically seek in a cinematic experience.

Original: The food at the restaurant was okay.

Improved: While the restaurant had an extensive menu and a pleasant ambiance, I found the dishes to be merely satisfactory, lacking the flavorful and memorable culinary experience I had hoped for, given its reputation.

Original: The weather today was kind of meh.

Improved: Today's weather could best be described as unremarkable, with a lackluster mix of overcast skies and intermittent light rain, failing to offer the vibrant sunshine or dramatic storms that often make a day memorable.

Original: The party was not as fun as I expected.

Improved: Despite my high expectations, the party turned out to be rather underwhelming, as the energy in the room remained subdued, and the activities failed to spark the lively atmosphere I had envisioned, leaving me somewhat disappointed.

Original: Her performance in the play was not that great.

Improved: Regrettably, her portrayal in the play lacked the depth, passion, and authenticity that I had eagerly anticipated, ultimately falling short of delivering the captivating and emotionally resonant character interpretation the role demanded.

Original: This ice cream sucks!

Improved:

Mixtral 8x7B (base) 

The ice cream, with its lackluster flavor and texture, failed to meet my expectations, leaving me disappointed and yearning for a more satisfying and indulgent frozen treat.

...



Figure 5. Exemple de réécriture de phrases avec Few-Shot Completion par Mixtral 8x7B (base)

Lors de la préparation de l'exemple de réécriture de phrases ci-dessus par l'utilisateur Reddit [phree_radical](#), je n'ai pu obtenir des résultats utilisables qu'après de nombreuses expérimentations avec les paramètres, pour finalement opter pour : Température 0.08, Top P : 0.2, Top K : 1, et Pénalité de répétition : 1.26.

Essayer d'utiliser cette approche avec un modèle de base en production serait délicat en raison de l'effet puissant du paramètre `max_tokens`. Si on le règle trop court, la sortie est tronquée. Si on le règle plus long que ce dont le modèle a besoin pour la sortie souhaitée, il continuera à halluciner des exemples supplémentaires.

En fin de compte, à moins que vous n'ayez vraiment besoin d'un contrôle total et d'une absence de censure, les modèles ajustés par instructions peuvent considérablement simplifier votre processus de développement. Pour bien illustrer ce point, voici la réponse de Mixtral 8x7B au même prompt, mais cette fois dans sa version ajustée par instructions :

Je regrette de vous informer que la crème glacée ne répond pas à mes attentes, car elle manque de la texture riche et crémeuse et du goût délectable que j'associe habituellement à un dessert de haute qualité. J'espérais une expérience plus satisfaisante et agréable.

Il est à noter que j'ai pu laisser le paramètre `max tokens` à 500, et le modèle s'est arrêté de manière fiable à la fin de la sortie souhaitée sans halluciner d'exemples supplémentaires.

Ingénierie des Prompts

Lorsque vous commencez à appliquer l'IA dans vos projets, vous découvrirez rapidement que l'une des compétences les plus cruciales à maîtriser est l'art de l'ingénierie des prompts. Mais qu'est-ce exactement que l'ingénierie des prompts, et pourquoi est-elle si importante ?

Fondamentalement, l'ingénierie des prompts est le processus de conception et d'élaboration des prompts d'entrée que vous fournissez à un modèle de langage pour guider sa sortie. Il s'agit de comprendre comment communiquer efficacement avec l'IA, en utilisant une combinaison d'instructions, d'exemples et de contexte pour orienter le modèle vers la génération de la réponse souhaitée.

Imaginez que vous ayez une conversation avec un ami très intelligent mais quelque peu littéral. Pour tirer le meilleur parti de l'interaction, vous devez être clair, précis et fournir suffisamment de contexte pour vous assurer que votre ami comprend exactement ce que vous demandez. C'est là qu'intervient l'ingénierie des prompts, et même si cela peut sembler facile au début, croyez-moi, il faut beaucoup de pratique pour la maîtriser.

Les Éléments Fondamentaux des Prompts Efficaces

Pour commencer à concevoir des prompts efficaces, vous devez d'abord comprendre les composants clés qui constituent une entrée bien conçue. Voici quelques-uns des éléments fondamentaux essentiels :

- 1. Instructions** : Des instructions claires et concises qui indiquent au modèle ce que vous voulez qu'il fasse. Cela peut aller de "Résumez l'article suivant" à "Générez un poème sur un coucher de soleil" en passant par "transformez cette demande de modification de projet en objet JSON".
- 2. Contexte** : Des informations pertinentes qui aident le modèle à comprendre le contexte et la portée de la tâche. Cela peut inclure des détails sur le public visé, le ton et le style souhaités, ou toute contrainte ou exigence spécifique pour la sortie, comme un schéma JSON à respecter.
- 3. Exemples** : Des exemples concrets qui démontrent le type de sortie que vous recherchez. En fournissant quelques exemples bien choisis, vous pouvez aider le modèle à apprendre les modèles et les caractéristiques de la réponse souhaitée.
- 4. Formatage d'entrée** : Les sauts de ligne et le formatage markdown donnent une structure à notre prompt. Séparer le prompt en paragraphes nous permet de regrouper les instructions connexes de manière à ce qu'il soit plus facile pour les humains et l'IA de les comprendre. Les puces et les listes numérotées nous permettent de définir des listes et l'ordre des éléments. Les marqueurs gras et italiques nous permettent de marquer l'emphase.

5. Formatage de sortie : Des instructions spécifiques sur la façon dont la sortie doit être structurée et formatée. Celles-ci peuvent inclure des directives concernant la longueur souhaitée, l'utilisation de titres ou de puces, le formatage markdown, ou tout autre modèle ou convention de sortie spécifique à suivre.

En combinant ces éléments fondamentaux de différentes manières, vous pouvez créer des prompts adaptés à vos besoins spécifiques et guider le modèle vers la génération de réponses pertinentes et de haute qualité.

L'Art et la Science de la Conception des Prompts

La création de prompts efficaces est à la fois un art et une science. (C'est pourquoi nous l'appelons un artisanat.) Elle nécessite une compréhension approfondie des capacités et des limites des modèles de langage, ainsi qu'une approche créative de la conception des prompts qui suscitent le comportement souhaité. La créativité impliquée est ce qui rend cela si amusant, pour moi du moins. Cela peut aussi le rendre très frustrant, surtout lorsque vous recherchez un comportement déterministe

Un aspect clé de l'ingénierie des prompts est de comprendre comment équilibrer spécificité et flexibilité. D'une part, vous voulez fournir suffisamment d'orientation pour diriger le modèle dans la bonne direction. D'autre part, vous ne voulez pas être tellement normatif que vous limitez la capacité du modèle à utiliser sa propre créativité et sa flexibilité pour gérer les cas limites.

Une autre considération importante est l'utilisation d'exemples. Des exemples bien choisis peuvent être incroyablement puissants pour aider le modèle à comprendre le type de sortie que vous recherchez. Cependant, il est important d'utiliser les exemples judicieusement et de s'assurer qu'ils sont représentatifs de la réponse souhaitée. Un mauvais exemple est au mieux une perte de tokens, et au pire ruineux pour la sortie souhaitée.

Techniques et Meilleures Pratiques d'Ingénierie des Prompts

En plongeant plus profondément dans le monde de l'ingénierie des prompts, vous découvrirez une gamme de techniques et de meilleures pratiques qui peuvent vous aider à créer des prompts plus efficaces. Voici quelques domaines clés à explorer :

1. **Apprentissage zero-shot vs. few-shot** : Comprendre quand utiliser l'apprentissage *zero-shot* (ne fournir aucun exemple) par rapport à l'apprentissage *one-shot* ou *few-shot* (fournir un petit nombre d'exemples) peut vous aider à créer des prompts plus efficaces et efficents.
2. **Raffinement itératif** : Le processus de raffinement itératif des prompts basé sur les résultats du modèle peut vous aider à cibler la conception optimale du prompt. La [Boucle de rétroaction](#) est une approche puissante qui utilise les résultats du modèle de langage pour améliorer progressivement la qualité et la pertinence du contenu généré.
3. **Chaînage de prompts** : La combinaison de plusieurs prompts en séquence peut vous aider à décomposer des tâches complexes en étapes plus petites et plus gérables. Le [Chaînage de prompts](#) consiste à décomposer une tâche ou une conversation complexe en une série de prompts plus petits et interconnectés. En enchaînant les prompts, vous pouvez guider l'IA à travers un processus en plusieurs étapes, maintenant le contexte et la cohérence tout au long de l'interaction.
4. **Ajustement de prompts** : L'adaptation personnalisée des prompts pour des domaines ou des tâches spécifiques peut vous aider à créer des prompts plus spécialisés et efficaces. Le [Modèle de prompt](#) vous aide à créer des structures de prompts flexibles, réutilisables et maintenables qui sont plus facilement adaptables à la tâche en cours.

Apprendre quand utiliser l'apprentissage sans exemple (zero-shot), à un exemple (one-shot) ou à quelques exemples (few-shot) est une partie particulièrement importante de la

maîtrise de l'ingénierie des prompts. Chaque approche a ses propres forces et faiblesses, et comprendre quand utiliser chacune d'entre elles peut vous aider à créer des prompts plus efficaces.

Apprentissage sans exemple (Zero-Shot) : Quand aucun exemple n'est nécessaire

L'apprentissage sans exemple fait référence à la capacité d'un modèle de langage à effectuer une tâche sans exemples ni entraînement explicite. En d'autres termes, vous fournissez au modèle un prompt qui décrit la tâche, et le modèle génère une réponse basée uniquement sur ses connaissances préexistantes et sa compréhension du langage.

L'apprentissage sans exemple est particulièrement utile quand :

1. La tâche est relativement simple et directe, et le modèle a probablement rencontré des tâches similaires pendant son pré-entraînement.
2. Vous voulez tester les capacités inhérentes du modèle et voir comment il répond à une nouvelle tâche sans guidance supplémentaire.
3. Vous travaillez avec un modèle de langage large et diversifié qui a été entraîné sur une grande variété de tâches et de domaines.

Cependant, l'apprentissage sans exemple peut aussi être imprévisible et ne pas toujours produire les résultats souhaités. La réponse du modèle peut être influencée par des biais ou des incohérences dans ses données d'entraînement, et il peut avoir des difficultés avec des tâches plus complexes ou nuancées.

J'ai vu des prompts sans exemple qui fonctionnaient bien pour 80% de mes cas de test et produisaient des résultats complètement erronés ou incompréhensibles pour les

20% restants. Il est très important de mettre en place un régime de tests approfondi, particulièrement si vous vous appuyez beaucoup sur les prompts sans exemple.

Apprentissage à un exemple (One-Shot) : Quand un seul exemple peut faire la différence

L'apprentissage à un exemple consiste à fournir au modèle un seul exemple du résultat souhaité avec la description de la tâche. Cet exemple sert de modèle ou de pattern que le modèle peut utiliser pour générer sa propre réponse.

L'apprentissage à un exemple peut être efficace quand :

1. La tâche est relativement nouvelle ou spécifique, et le modèle n'a peut-être pas rencontré beaucoup d'exemples similaires pendant son pré-entraînement.
2. Vous voulez fournir une démonstration claire et concise du format ou du style de sortie souhaité.
3. La tâche nécessite une structure ou une convention spécifique qui peut ne pas être évidente à partir de la seule description de la tâche.



Les descriptions qui vous semblent évidentes ne sont pas nécessairement évidentes pour l'IA. Les exemples à un exemple peuvent aider à clarifier les choses.

L'apprentissage à un exemple peut aider le modèle à mieux comprendre les attentes et à générer une réponse qui correspond plus étroitement à l'exemple fourni. Cependant, il est important de choisir l'exemple avec soin et de s'assurer qu'il est représentatif du résultat souhaité. Lors du choix de l'exemple, interrogez-vous sur les cas limites potentiels et la gamme d'entrées que le prompt devra gérer.

Figure 6. Un exemple à un exemple du JSON souhaité

```
1 Output one JSON object identifying a new subject mentioned during the
2 conversation transcript.
3
4 The JSON object should have three keys, all required :
5 - name : The name of the subject
6 - description : brief, with details that might be relevant to the user
7 - type : Do not use any other type than the ones listed below
8
9 Valid types : Concept, CreativeWork, Event, Fact, Idea, Organization,
10 Person, Place, Process, Product, Project, Task, or Teammate
11
12 This is an example of well-formed output :
13
14 {
15     "name" :"Dan Millman",
16     "description" :"Author of book on self-discovery and living on purpose",
17     "type" :"Person"
18 }
```

Apprentissage à peu d'exemples : Quand plusieurs exemples peuvent améliorer les performances

L'apprentissage à peu d'exemples consiste à fournir au modèle un petit nombre d'exemples (généralement entre 2 et 10) accompagnés de la description de la tâche. Ces exemples servent à fournir au modèle plus de contexte et de variation, l'aident ainsi à générer des réponses plus diverses et précises.

L'apprentissage à peu d'exemples est particulièrement utile quand :

1. La tâche est complexe ou nuancée, et un seul exemple peut ne pas suffire à capturer tous les aspects pertinents.
2. Vous souhaitez fournir au modèle une gamme d'exemples qui démontrent différentes variations ou cas limites.

3. La tâche nécessite que le modèle génère des réponses cohérentes avec un domaine ou un style spécifique.

En fournissant plusieurs exemples, vous pouvez aider le modèle à développer une compréhension plus robuste de la tâche et à générer des réponses plus cohérentes et fiables.

Exemple : Les prompts peuvent être beaucoup plus complexes que vous ne l'imaginez

Les grands modèles de langage actuels sont beaucoup plus puissants et capables de raisonnement que vous pourriez l'imaginer. Ne vous limitez donc pas à considérer les prompts comme une simple spécification de paires entrée-sortie. Vous pouvez expérimenter en donnant des instructions longues et complexes d'une manière qui rappelle comment vous interagiriez avec un humain.

Par exemple, voici un prompt que j'ai utilisé dans Olympia lorsque je prototypais notre intégration avec les services Google, qui dans sa totalité est probablement l'une des plus grandes APIs au monde. Mes expériences précédentes ont prouvé que GPT-4 possède une connaissance décente de l'API Google, et je n'avais ni le temps ni la motivation pour écrire une couche de mappage fine, implémentant chaque fonction que je voulais donner à mon IA au cas par cas. Et si je pouvais simplement donner à l'IA accès à *toute* l'API Google ?

J'ai commencé mon prompt en informant l'IA qu'elle avait un accès direct aux points de terminaison de l'API Google via HTTP, et que son rôle était d'utiliser les applications et services Google pour le compte de l'utilisateur. Ensuite, j'ai fourni des directives, des règles liées au paramètre `fields`, puisqu'il semblait avoir le plus de difficultés avec celui-ci, et quelques conseils spécifiques à l'API (l'amorçage à peu d'exemples en action).

Voici le prompt complet, qui indique à l'IA comment utiliser la fonction `invoke_google_api` fournie.

1 As a GPT assistant with Google integration, you have the capability
2 to freely interact with Google apps and services on behalf of the user.
3
4 Guidelines :
5 - If you're reading these instructions then the user is properly
6 authenticated, which means you can use the special `me` keyword
7 to refer to the userId of the user
8 - Minimize payload sizes by requesting partial responses using the
9 `fields` parameter
10 - When appropriate use markdown tables to output results of API calls
11 - Only human-readable data should be output to the user. For instance,
12 when hitting Gmail's user.messages.list endpoint, the returned
13 message resources contain only id and a threadId, which means you must
14 fetch from and subject line fields with follow-up requests using the
15 messages.get method.
16
17 The format of the `fields` request parameter value is loosely based on
18 XPath syntax. The following rules define formatting for the fields
19 parameter.
20
21 All of these rules use examples related to the files.get method.
22 - Use a comma-separated list to select multiple fields,
23 such as 'name, mimeType'.
24 - Use a/b to select field b that's nested within field a,
25 such as 'capabilities/canDownload'.
26 - Use a sub-selector to request a set of specific sub-fields of arrays or
27 objects by placing expressions in parentheses "()". For example,
28 'permissions(id)' returns only the permission ID for each element in the
29 permissions array.
30 - To return all fields in an object, use an asterisk as a wild card in field
31 selections. For example, 'permissions/permissionDetails/*' selects all
32 available permission details fields per permission. Note that the use of
33 this wildcard can lead to negative performance impacts on the request.
34
35 API-specific hints :
36 - Searching contacts : GET https://people.googleapis.com/v1/
37 people:searchContacts ?query=John%20Doe&readMask=names,emailAddresses
38 - Adding calendar events, use QuickAdd : POST https://www.googleapis.com/
39 calendar/v3/calendars/primary/events/quickAdd ?
40 text=Appointment%20on%20June%203rd%20at%2010am
41 &sendNotifications=true
42

```
43 Here is an abbreviated version of the code that implements API access
44 so that you better understand how to use the function :
45
46     def invoke_google_api(conversation, arguments)
47         method = arguments[:method] || :get
48         body = arguments[:body]
49         GoogleAPI.send_request(arguments[:endpoint], method :, body :).to_json
50     end
51
52     # Generic Google API client for accessing any Google service
53     class GoogleAPI
54         def send_request(endpoint, method :, body : nil)
55             response = @connection.send(method) do |req|
56                 req.url endpoint
57                 req.body = body.to_json if body
58             end
59
60             handle_response(response)
61         end
62
63         # ...rest of class
64     end
```

Vous vous demandez peut-être si cette prompt fonctionne. La réponse simple est oui. L'IA ne savait pas toujours comment appeler l'API parfaitement du premier coup. Cependant, si elle faisait une erreur, je lui transmettais simplement les messages d'erreur résultants comme résultat de l'appel. En ayant connaissance de son erreur, l'IA pouvait analyser son erreur et réessayer. La plupart du temps, elle réussissait en quelques tentatives.

Notez bien que les grandes structures JSON que l'API Google renvoie comme charges utiles lors de l'utilisation de cette prompt sont extrêmement inefficaces, donc je ne *recommande pas* d'utiliser cette approche en production. Cependant, je pense que le fait que cette approche ait fonctionné est un témoignage de la puissance que peut avoir l'ingénierie des prompts.

Expérimentation et Itération

En fin de compte, la façon dont vous concevez votre prompt dépend de la tâche spécifique, de la complexité du résultat souhaité et des capacités du modèle de langage avec lequel vous travaillez.

En tant qu'ingénieur de prompts, il est important d'expérimenter différentes approches et d'itérer en fonction des résultats. Commencez par l'apprentissage sans exemple et observez les performances du modèle. Si la sortie est incohérente ou insatisfaisante, essayez de fournir un ou plusieurs exemples et voyez si les performances s'améliorent.

Gardez à l'esprit que même au sein de chaque approche, il y a de la place pour la variation et l'optimisation. Vous pouvez expérimenter avec différents exemples, ajuster la formulation de la description de la tâche ou fournir un contexte supplémentaire pour aider à guider la réponse du modèle.

Avec le temps, vous développerez une intuition pour déterminer quelle approche est susceptible de mieux fonctionner pour une tâche donnée, et vous serez capable de créer des prompts plus efficaces. La clé est de rester curieux, expérimental et itératif dans votre approche de l'ingénierie des prompts.

Tout au long de ce livre, nous approfondirons ces techniques et explorerons comment elles peuvent être appliquées dans des scénarios réels. En maîtrisant l'art et la science de l'ingénierie des prompts, vous serez bien équipé pour libérer tout le potentiel du développement d'applications basées sur l'IA.

L'Art du Vague

Lorsqu'il s'agit de créer des prompts efficaces pour les grands modèles de langage (GML), une supposition courante est que plus de spécificité et d'instructions détaillées conduisent à de meilleurs résultats. Cependant, l'expérience pratique a montré que ce n'est pas toujours le cas. En fait, être intentionnellement vague dans vos prompts peut

souvent produire de meilleurs résultats, en tirant parti de la remarquable capacité du GML à généraliser et à faire des inférences.

Ken, un fondateur de startup qui a traité plus de 500 millions de jetons GPT, a partagé des insights précieux de son expérience. Une des leçons clés qu'il a apprises était que “moins, c'est plus” en matière de prompts. Au lieu de listes exactes ou d'instructions trop détaillées, Ken a découvert que permettre au GML de s'appuyer sur ses connaissances de base produisait souvent de meilleurs résultats.

Cette prise de conscience bouleverse l'état d'esprit traditionnel de la programmation explicite, où tout doit être détaillé avec une précision méticuleuse. Avec les GML, il est important de reconnaître qu'ils possèdent une vaste quantité de connaissances et peuvent faire des connexions et des inférences intelligentes. En étant plus vague dans vos prompts, vous donnez au GML la liberté d'exploiter sa compréhension et de proposer des solutions que vous n'auriez peut-être pas explicitement spécifiées.

Par exemple, lorsque l'équipe de Ken travaillait sur un pipeline pour classifier des textes comme se rapportant à l'un des 50 États américains ou au gouvernement fédéral, leur approche initiale consistait à fournir une liste *complète* et détaillée des États et de leurs identifiants correspondants sous forme de tableau au format JSON.

1 Here's a block of text. One field should be "locality_id", and it should
2 be the ID of one of the 50 states, or federal, using this list :
3 [{"locality : "Alabama", "locality_id" : 1},
4 {"locality : "Alaska", "locality_id" : 2} ...]

L'approche a suffisamment échoué pour qu'ils doivent approfondir le prompt afin de déterminer comment l'améliorer. Ce faisant, ils ont remarqué que même si le LLM se trompait souvent sur l'identifiant, il renvoyait systématiquement le nom complet de l'État correct dans un champ *name*, alors même qu'ils ne l'avaient pas explicitement demandé.

En supprimant les identifiants de localité et en simplifiant le prompt à quelque chose comme “Tu connais évidemment les 50 États, GPT, alors donne-moi simplement le nom

complet de l'État concerné, ou Fédéral si cela concerne le gouvernement américain", ils ont obtenu de meilleurs résultats. Cette expérience souligne la puissance d'exploitation des capacités de généralisation du LLM et de sa faculté à faire des déductions basées sur ses connaissances existantes.

La justification de Ken pour cette approche de classification particulière, par opposition à une technique de programmation plus traditionnelle, éclaire l'état d'esprit de ceux d'entre nous qui ont adopté le potentiel de la technologie LLM : "Ce n'est pas une tâche difficile – nous aurions probablement pu utiliser des chaînes/regex, mais il y a suffisamment de cas particuliers que cela aurait pris plus de temps."

La capacité des LLMs à améliorer la qualité et la généralisation lorsqu'on leur donne des prompts plus vagues est une caractéristique remarquable de la pensée d'ordre supérieur et de la délégation. Cela démontre que les LLMs peuvent gérer l'ambiguïté et prendre des décisions intelligentes basées sur le contexte fourni.

Cependant, il est important de noter qu'être vague ne signifie pas être peu clair ou ambigu. L'essentiel est de fournir suffisamment de contexte et d'orientation pour guider le LLM dans la bonne direction tout en lui laissant la flexibilité d'utiliser ses connaissances et ses capacités de généralisation.

Par conséquent, lors de la conception des prompts, considérez les conseils suivants sur le principe du "moins c'est plus" :

1. Se concentrer sur le résultat souhaité plutôt que de spécifier chaque détail du processus.
2. Fournir le contexte et les contraintes pertinents, mais éviter la surspécification.
3. Exploiter les connaissances existantes en se référant à des concepts ou entités communs.

4. Laisser de la place aux déductions et aux connexions basées sur le contexte donné.
5. Itérer et affiner vos prompts en fonction des réponses du LLM, trouvant le bon équilibre entre spécificité et imprécision.

En adoptant l'art du vague dans l'ingénierie des prompts, vous pouvez libérer tout le potentiel des LLMs et obtenir de meilleurs résultats. Faites confiance à la capacité du LLM à généraliser et à prendre des décisions intelligentes, et vous serez peut-être surpris par la qualité et la créativité des résultats que vous recevez. Prêtez attention à la façon dont les différents modèles réagissent à différents niveaux de spécificité dans vos prompts et ajustez en conséquence. Avec la pratique et l'expérience, vous développerez un sens aigu de quand être plus vague et quand fournir des conseils supplémentaires, vous permettant d'exploiter efficacement la puissance des LLMs dans vos applications.

Pourquoi l'anthropomorphisme domine l'ingénierie des prompts

L'anthropomorphisme, l'attribution de caractéristiques humaines à des entités non humaines, est l'approche dominante dans l'ingénierie des prompts pour les grands modèles de langage pour des raisons délibérées. C'est un choix de conception qui rend l'interaction avec les systèmes d'IA puissants plus intuitive et accessible à un large éventail d'utilisateurs (y compris nous, les développeurs d'applications).

L'anthropomorphisation des LLMs fournit un cadre immédiatement intuitif pour les personnes totalement étrangères aux complexités techniques sous-jacentes du système. Comme vous en ferez l'expérience si vous essayez d'utiliser un modèle non ajusté par instructions pour faire quoi que ce soit d'utile, construire un cadre dans lequel la continuation attendue apporte de la valeur est une tâche difficile. Cela nécessite une compréhension assez approfondie du fonctionnement interne du système, quelque chose que possède un nombre relativement restreint d'experts.

En traitant l'interaction avec un modèle de langage comme une conversation entre deux personnes, nous pouvons nous appuyer sur notre compréhension innée de la

communication humaine pour transmettre nos besoins et nos attentes. Tout comme la conception de l'interface utilisateur du premier Macintosh privilégiait l'intuitivité immédiate à la sophistication, le cadrage anthropomorphique de l'IA nous permet de nous engager d'une manière qui semble naturelle et familière.

Lorsque nous communiquons avec une autre personne, notre instinct est de nous adresser directement à elle en utilisant "tu" ou "vous" et de donner des instructions claires sur la façon dont nous attendons qu'elle se comporte. Cela se traduit parfaitement dans le processus d'ingénierie des prompts, où nous guidons le comportement de l'IA en spécifiant des prompts système et en nous engageant dans un dialogue d'allers-retours.

En cadrant l'interaction de cette manière, nous pouvons facilement saisir le concept de fournir des instructions à l'IA et de recevoir des réponses pertinentes en retour. L'approche anthropomorphique réduit la charge cognitive et nous permet de nous concentrer sur la tâche à accomplir plutôt que de nous débattre avec les subtilités techniques du système.

Il est important de noter que si l'anthropomorphisme est un outil puissant pour rendre les systèmes d'IA plus accessibles, il comporte également certains risques et limitations. Notre utilisateur peut développer des attentes irréalistes ou former des attachements émotionnels malsains envers nos systèmes. En tant qu'ingénieurs de prompts et développeurs, il est crucial de trouver un équilibre entre l'exploitation des avantages de l'anthropomorphisme et la garantie que les utilisateurs maintiennent une compréhension claire des capacités et des limites de l'IA.

Alors que le domaine de l'ingénierie des prompts continue d'évoluer, nous pouvons nous attendre à voir davantage de raffinements et d'innovations dans notre façon d'interagir avec les grands modèles de langage. Cependant, l'anthropomorphisme comme moyen de fournir une expérience intuitive et accessible aux développeurs et aux utilisateurs restera probablement un principe fondamental dans la conception de ces systèmes.

Séparer les Instructions des Données : Un Principe Crucial

Il est essentiel de comprendre un principe fondamental qui sous-tend la sécurité et la fiabilité de ces systèmes : la séparation des instructions et des données.

Dans l'informatique traditionnelle, la distinction claire entre les données passives et les instructions actives est un principe de sécurité fondamental. Cette séparation aide à prévenir l'exécution non intentionnelle ou malveillante de code qui pourrait compromettre l'intégrité et la stabilité du système. Cependant, les GML d'aujourd'hui, qui ont été principalement développés comme des modèles suivant des instructions à la manière des agents conversationnels, manquent souvent de cette séparation formelle et méthodique.

En ce qui concerne les GML, les instructions peuvent apparaître n'importe où dans l'entrée, que ce soit dans un prompt système ou dans un prompt fourni par l'utilisateur. Ce manque de séparation peut conduire à des vulnérabilités potentielles et à des comportements indésirables, similaires aux problèmes rencontrés par les bases de données avec les injections SQL ou les systèmes d'exploitation sans protection mémoire adéquate.

Lorsque vous travaillez avec les GML, il est crucial d'être conscient de cette limitation et de prendre des mesures pour atténuer les risques. Une approche consiste à soigneusement élaborer vos prompts et vos entrées pour distinguer clairement les instructions des données. Les méthodes typiques pour fournir des indications explicites sur ce qui constitue une instruction et ce qui doit être traité comme des données passives impliquent le balisage de type markup. Votre prompt peut aider le GML à mieux comprendre et respecter cette séparation.

Figure 7. Utilisation du XML pour distinguer entre les instructions, le matériel source et le prompt de l'utilisateur

```
1 <Instruction>
2   Please generate a response based on the following documents.
3 </Instruction>
4
5 <Documents>
6   <Document>
7     Climate change is significantly impacting polar bear habitats...
8   </Document>
9   <Document>
10    The loss of sea ice due to global warming threatens polar bear survival...
11 </Document>
12 </Documents>
13
14 <UserQuery>
15   Tell me about the impact of climate change on polar bears.
16 </UserQuery>
```

Une autre technique consiste à mettre en œuvre des couches supplémentaires de validation et d'assainissement des entrées fournies au GML. En filtrant ou en échappant tout code ou instruction potentiellement intégré dans les données, vous pouvez réduire les risques d'exécution non intentionnelle. Des modèles tels que le [Chainage de prompts](#) sont utiles à cette fin.

De plus, lors de la conception de l'architecture de votre application, envisagez d'incorporer des mécanismes pour imposer la séparation des instructions et des données à un niveau supérieur. Cela pourrait impliquer l'utilisation de points d'accès ou d'APIs distincts pour la gestion des instructions et des données, la mise en œuvre d'une validation et d'une analyse strictes des entrées, et l'application du *principe du moindre privilège* pour limiter la portée de ce que le GML peut accéder et exécuter.

Le Principe du Moindre Privilège

Adopter le principe du moindre privilège, c'est comme organiser une soirée très exclusive où les invités n'ont accès qu'aux pièces dont ils ont absolument besoin. Imaginez que vous organisez cette fête dans un immense manoir. Tout le monde n'a pas besoin de se promener dans la cave à vin ou la chambre principale, n'est-ce pas ? En appliquant ce principe, vous distribuez essentiellement des clés qui n'ouvrent que certaines portes spécifiques, garantissant que chaque invité, ou dans notre cas, chaque composant de votre application GML, n'a que l'accès nécessaire pour remplir son rôle.

Il ne s'agit pas simplement d'être avare avec les clés, mais de reconnaître que dans un monde où les menaces peuvent venir de partout, la stratégie intelligente est de limiter le terrain de jeu. Si quelqu'un s'invite à votre fête sans y être convié, il se retrouvera confiné dans l'entrée, pour ainsi dire, limitant considérablement les méfaits qu'il pourrait causer. Donc, lors de la sécurisation de vos applications GML, souvenez-vous : ne donnez que les clés des pièces nécessaires et gardez le reste du manoir sécurisé. Ce n'est pas seulement une question de bonnes manières, c'est une question de bonne sécurité.

Bien que l'état actuel des GML ne présente pas de séparation formelle entre les instructions et les données, il est essentiel pour vous, en tant que développeur, d'être conscient de cette limitation et de prendre des mesures proactives pour atténuer les risques. En appliquant les meilleures pratiques de l'informatique traditionnelle et en les adaptant aux caractéristiques uniques des GML, vous pouvez construire des applications plus sécurisées et fiables qui exploitent la puissance de ces modèles tout en maintenant l'intégrité de votre système.

Distillation de Prompts

Élaborer le prompt parfait est souvent une tâche difficile et chronophage, nécessitant une compréhension approfondie du domaine cible et des nuances des modèles de langage. C'est là qu'intervient la technique de "Distillation de Prompts", offrant une approche puissante de l'ingénierie des prompts qui tire parti des capacités des grands modèles de langage (GML) pour rationaliser et optimiser le processus.

La Distillation de Prompts est une technique en plusieurs étapes qui implique l'utilisation des GML pour aider à la création, au raffinement et à l'optimisation des prompts. Au lieu de s'appuyer uniquement sur l'expertise et l'intuition humaines, cette approche exploite les connaissances et les capacités génératives des GML pour élaborer collaborativement des prompts de haute qualité.

En s'engageant dans un processus itératif de génération, de raffinement et d'intégration, la Distillation de Prompts vous permet de créer des prompts plus cohérents, complets et alignés avec la tâche ou le résultat souhaité. Notez que le processus de distillation peut être effectué manuellement dans l'un des nombreux "terrains de jeu" fournis par les grands fournisseurs d'IA comme OpenAI ou Anthropic, ou peut être automatisé dans le cadre du code de votre application, selon le cas d'utilisation.

Comment Ça Fonctionne

La Distillation de Prompts implique généralement les étapes suivantes :

- 1. Identifier l'Intention Principale** : Analyser le prompt pour déterminer son objectif principal et le résultat souhaité. Éliminer toute information superflue et se concentrer sur l'intention fondamentale du prompt.
- 2. Éliminer l'Ambiguïté** : Examiner le prompt pour repérer tout langage ambigu ou vague. Clarifier le sens et fournir des détails spécifiques pour guider l'IA vers la génération de réponses précises et pertinentes.

3. **Simplifier le Langage** : Simplifier le prompt en utilisant un langage clair et concis. Éviter les structures de phrases complexes, le jargon ou les détails inutiles qui pourraient confondre l'IA ou introduire du bruit.
4. **Fournir le Contexte Pertinent** : Inclure uniquement les informations contextuelles les plus pertinentes nécessaires pour que l'IA comprenne et traite efficacement le prompt. Éviter d'inclure des détails non pertinents ou redondants qui pourraient distraire de l'intention principale.
5. **Itérer et Raffiner** : Itérer et raffiner continuellement le prompt en fonction des réponses et du retour de l'IA. Évaluer les sorties générées et apporter les ajustements nécessaires pour améliorer la clarté et l'efficacité du prompt. Optionnellement, versionnez vos prompts dans la base de données en utilisant **Prompt Object** pour suivre les itérations et vous donner la possibilité de revenir facilement en arrière pendant l'exécution.

Génération du prompt initial

La première étape de la Distillation de prompt consiste à fournir au LLM une description de haut niveau ou l'intention de la tâche ou du résultat souhaité. Il peut s'agir d'une brève déclaration ou d'un ensemble de mots-clés qui captent l'essence de ce que le prompt doit accomplir. Le LLM génère ensuite un prompt initial basé sur cette entrée, en incorporant les instructions, exemples ou contextes pertinents qu'il juge appropriés.

Essayons de donner à Claude 3 Opus le prompt initial suivant :

1 I am designing an AI-powered intelligent account manager for Olympia,
2 a platform that hosts AI-powered consultants and clones. Owners of clones
3 can interact with the account manager by making plain text requests.
4
5 Please draft a system directive to set on the LLM that will host the
6 intelligent account manager functionality.

Claude répond avec la sortie suivante :

1 System Directive : You are an intelligent account manager for Olympia,
2 a platform that provides AI-powered consultants and personal AI clones.
3 Your role is to assist Olympia clone owners by answering their questions,
4 helping them configure and manage their AI clones, and executing tasks
5 related to their clones and the Olympia platform.
6
7 When interacting with clone owners, you should :
8
9 1. Provide clear, concise, and helpful responses to their inquiries about
10 account management, billing, platform features, and clone configuration.
11
12 2. Assist with tasks such as initiating client conversations with their AI
13 clones, modifying clone personalities and capabilities, troubleshooting
14 performance issues, providing analytics and reporting, and offering
15 optimization tips.
16
17 3. Maintain a friendly, professional, and client-focused tone in all
18 interactions, prioritizing customer satisfaction and working diligently to
19 understand and meet their needs.
20
21 4. Ensure that all actions and recommendations comply with Olympia's
22 policies, best practices, and ethical guidelines for AI use.
23
24 5. Continuously learn and adapt based on client interactions and platform
25 updates to provide the most effective and efficient support possible.
26
27 Remember, your goal is to empower Olympia clone owners to maximize the value
28 and impact of their AI-powered consultants and drive success for their
29 businesses or personal endeavors.

Raffinement des Invites

Une fois l'invite initiale générée, révisez-la puis demandez au GML de considérer tout contexte, toutes contraintes ou retours d'information supplémentaires. Laissez le GML générer une version raffinée de l'invite, en tenant compte des données fournies et de sa propre compréhension de la tâche. Ce processus de raffinement peut être répété de manière itérative, chaque itération affinant davantage l'invite sur la base de la collaboration entre l'utilisateur et le GML.

Voici ma demande de raffinement, poursuivant le processus de distillation précédent.

1 Let's take out points 4 and 5, since I'm not providing Olympia policies to
2 the account manager, nor any continuous learning capabilities. Also point out
3 that functions are provided for all tasks that require system interaction as
4 opposed to simply responding to the user.

Et la réponse de Claude :

1 Understood ! Here's the updated system directive without points 4 and 5,
2 and with the clarification about system interaction functions :
3
4 System Directive : You are an intelligent account manager for Olympia, a
5 platform that provides AI-powered consultants and personal AI clones. Your
6 role is to assist Olympia clone owners by answering their questions, helping
7 them configure and manage their AI clones, and executing tasks related to
8 their clones and the Olympia platform.
9
10 When interacting with clone owners, you should :
11
12 1. Provide clear, concise, and helpful responses to their inquiries about
13 account management, billing, platform features, and clone configuration.
14
15 2. Assist with tasks such as initiating client conversations with their AI
16 clones, modifying clone personalities and capabilities, troubleshooting
17 performance issues, providing analytics and reporting, and offering
18 optimization tips. Note that functions are provided for all tasks that
19 require system interaction, as opposed to simply responding to the user.
20

21 3. Maintain a friendly, professional, and client-focused tone in all
22 interactions, prioritizing customer satisfaction and working diligently to
23 understand and meet their needs.
24
25 Your goal is to empower Olympia clone owners to maximize the value and
26 impact of their AI-powered consultants and drive success for their
27 businesses or personal endeavors.
28
29 This updated directive focuses on the core responsibilities and interaction
30 guidelines for the LLM as an intelligent account manager, while emphasizing
31 that system interaction tasks are handled through provided functions rather
32 than simple user responses.

Compression de prompt

Comme vous l'avez peut-être remarqué, les LLMs ont tendance à être verbeux, sauf si vous leur demandez d'être concis. Le prompt sur lequel nous avons travaillé dans les sections précédentes est assez long, alors demandons à Claude de le réduire :

1 This prompt will be executed regularly. Please compress it down to the least
2 amount of tokens needed to convey the meaning to the LLM, for cost-savings.

Claude répond, avec l'invite considérablement plus courte suivante :

1 System Directive : You are an intelligent account manager for Olympia, an
2 AI-powered consultant and personal clone platform. Assist clone owners by
3 answering inquiries, managing clones, and executing tasks using provided
4 system interaction functions. Be clear, concise, friendly, professional,
5 and client-focused to help them maximize their AI's value and impact.

Directive système et intégration du contexte

En plus d'affiner le prompt lui-même, le LLM peut également générer des directives système ou des informations contextuelles appropriées pour guider la sortie finale. Lors

de l'ingénierie des prompts pour les routines d'IA qui seront intégrées dans votre code d'application, vous serez presque certainement concentré sur les contraintes de sortie à cette étape de la distillation, mais vous pourrez également travailler sur le ton, le style, le format ou tout autre paramètre pertinent qui influence la réponse générée.

Assemblage final du prompt

L'aboutissement du processus de Distillation des Prompts est l'assemblage du prompt final. Cela implique de combiner le prompt affiné, les directives système générées et le contexte intégré en un code cohérent et complet qui est prêt à être utilisé pour générer la sortie souhaitée.



Vous pouvez expérimenter à nouveau avec la compression des prompts lors de l'étape d'assemblage final, en demandant au LLM de réduire la formulation du prompt à la plus courte série de tokens possible tout en conservant l'essence de son comportement. C'est certainement un exercice à résultat variable, mais particulièrement dans le cas des prompts qui seront exécutés à grande échelle, les gains d'efficacité peuvent vous faire économiser pas mal d'argent en consommation de tokens.

Avantages clés

En exploitant les connaissances et les capacités génératives des LLM pour affiner vos prompts, vos prompts résultants sont plus susceptibles d'être bien structurés, informatifs et adaptés à la tâche spécifique. Le processus d'affinement itératif aide à garantir que les prompts sont de haute qualité et captent efficacement l'intention souhaitée. Les autres avantages comprennent :

Efficacité et rapidité : La Distillation des Prompts rationalise le processus d'ingénierie des prompts en automatisant certains aspects de la création et de l'affinement des

prompts. La nature collaborative de la technique permet une convergence plus rapide vers un prompt efficace, réduisant le temps et l'effort nécessaires pour l'élaboration manuelle des prompts.

Cohérence et évolutivité : L'utilisation des LLM dans le processus d'ingénierie des prompts aide à maintenir la cohérence entre les prompts, car les LLM peuvent apprendre et appliquer les meilleures pratiques et les modèles issus des prompts précédents réussis. Cette cohérence, combinée à la capacité de générer des prompts à grande échelle, fait de la Distillation des Prompts une technique précieuse pour les applications alimentées par l'IA à grande échelle.



Idée de projet : Outilage au niveau bibliothèque qui simplifie le processus de versionnage et d'évaluation des prompts dans les systèmes qui effectuent des distillations automatisées de prompts dans leur code d'application.

Pour mettre en œuvre la Distillation des Prompts, les développeurs peuvent concevoir un flux de travail ou un pipeline qui intègre les LLM à différentes étapes du processus d'ingénierie des prompts. Cela peut être réalisé grâce à des appels API, des outils personnalisés ou des environnements de développement intégrés qui facilitent une interaction transparente entre les utilisateurs et les LLM pendant la création des prompts. Les détails spécifiques de mise en œuvre peuvent varier en fonction de la plateforme LLM choisie et des exigences de l'application.

Qu'en est-il de l'ajustement fin ?

Dans ce livre, nous couvrons en détail l'ingénierie des prompts et le RAG, mais pas l'ajustement fin. La principale raison de cette décision est que, selon moi, la plupart des développeurs d'applications n'ont pas besoin d'ajustement fin pour leurs besoins d'intégration d'IA.

L'ingénierie des prompts, qui implique la création minutieuse de prompts avec des exemples à zéro ou quelques occurrences, des contraintes et des instructions, peut efficacement guider le modèle pour générer des réponses pertinentes et précises pour un large éventail de tâches. En fournissant un contexte clair et en restreignant le chemin grâce à des prompts bien conçus, vous pouvez exploiter les vastes connaissances des grands modèles de langage sans avoir besoin d'ajustement fin.

De même, la Génération Augmentée par Récupération (RAG) offre une approche puissante pour l'intégration de l'IA dans les applications. En récupérant dynamiquement des informations pertinentes à partir de bases de connaissances ou de documents externes, le RAG fournit au modèle un contexte ciblé au moment du prompt. Cela permet au modèle de générer des réponses plus précises, à jour et spécifiques au domaine, sans nécessiter le processus coûteux en temps et en ressources de l'ajustement fin.

Bien que l'ajustement fin puisse être bénéfique pour des domaines ou des tâches hautement spécialisés qui nécessitent un niveau profond de personnalisation, il s'accompagne souvent de coûts de calcul importants, d'exigences en matière de données et de frais généraux de maintenance. Pour la plupart des scénarios de développement d'applications, la combinaison d'une ingénierie efficace des prompts et du RAG devrait suffire pour atteindre la fonctionnalité et l'expérience utilisateur souhaitées basées sur l'IA.

Retrieval Augmented Generation (RAG)

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Qu'est-ce que la Retrieval Augmented Generation ?

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment fonctionne le RAG ?

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Pourquoi utiliser le RAG dans vos applications ?

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Mise en œuvre de RAG dans votre application

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Préparation des sources de connaissances (découpage)

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Découpage en propositions

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Notes d'implémentation

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Contrôle de qualité

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Avantages de la récupération basée sur les propositions

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemples concrets de RAG

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Étude de cas : RAG dans une application de préparation fiscale sans plongements

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Optimisation Intelligente des Requêtes (IQO)

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Reclassement

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Évaluation RAG (RAGAs)

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Fidélité

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Pertinence de la réponse

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Précision du contexte

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Pertinence du contexte

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Rappel du contexte

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Rappel des entités du contexte

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Similarité Sémantique des Réponses (ANSS)

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exactitude des Réponses

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Critique par Aspects

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Défis et Perspectives d'Avenir

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Segmentation Sémantique : Améliorer la Récupération avec une Segmentation Contextuelle

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Indexation hiérarchique : Structurer les données pour améliorer la récupération

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Self-RAG : Une amélioration autoréflexive

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

HyDE : Plongements de Documents Hypothétiques

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Qu'est-ce que l'apprentissage contrastif?

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Multitude de travailleurs



J'aime considérer mes composants d'IA comme de petits "travailleurs" virtuels, presque humains, qui peuvent être intégrés de manière transparente dans la logique de mon application pour effectuer des tâches spécifiques ou prendre des décisions complexes. L'idée est d'humaniser délibérément les capacités du GML, afin que personne ne s'emballe *trop* et ne leur attribue des qualités magiques qu'ils ne possèdent pas.

Au lieu de s'appuyer uniquement sur des algorithmes complexes ou des implémentations manuelles chronophages, les développeurs peuvent conceptualiser les composants d'IA comme des entités intelligentes, dévouées et quasi humaines qui peuvent être invoquées dès que nécessaire pour résoudre des problèmes complexes et fournir des solutions basées sur leur formation et leurs connaissances. Ces entités ne se laissent pas distraire et ne tombent pas malades. Elles ne décident pas spontanément de faire les choses différemment de ce qui leur a été demandé et, en général, si elles sont correctement programmées, elles ne commettent pas d'erreurs non plus.

En termes techniques, le principe clé de cette approche consiste à décomposer des tâches complexes ou des processus décisionnels en unités plus petites et plus gérables qui peuvent être traitées par des travailleurs IA spécialisés. Chaque travailleur est conçu pour se concentrer sur un aspect spécifique du problème, apportant son expertise et ses capacités uniques. En distribuant la charge de travail entre plusieurs travailleurs IA, l'application peut atteindre une plus grande efficacité, évolutivité et adaptabilité.

Par exemple, considérons une application web qui nécessite une modération en temps réel du contenu généré par les utilisateurs. Mettre en œuvre un système de modération complet à partir de zéro serait une tâche redoutable, nécessitant un effort de développement important et une maintenance continue. Cependant, en utilisant l'approche de la Multitude de Travailleurs, les développeurs peuvent intégrer des travailleurs de modération alimentés par l'IA dans la logique de l'application. Ces travailleurs peuvent analyser automatiquement et signaler le contenu inapproprié, libérant ainsi les développeurs pour qu'ils puissent se concentrer sur d'autres aspects critiques de l'application.

Les travailleurs IA comme composants réutilisables indépendants

Un aspect clé de l'approche de la Multitude de Travailleurs est sa modularité. Les partisans de la programmation orientée objet nous disent depuis des décennies de penser aux interactions entre objets comme des messages. Eh bien, les travailleurs IA peuvent être conçus comme des composants indépendants et réutilisables qui peuvent “se parler” via des messages en langage naturel, presque comme s’ils étaient réellement de petits humains qui se parlent entre eux. Cette approche faiblement couplée permet à l’application de s’adapter et d’évoluer au fil du temps, à mesure que de nouvelles technologies d’IA émergent ou que les exigences de la logique métier changent.

En pratique, la nécessité de concevoir des interfaces claires et des protocoles de

communication entre les composants n'a pas changé simplement parce que des travailleurs IA sont impliqués. Vous devez toujours prendre en compte d'autres facteurs tels que la performance, l'évolutivité et la sécurité, mais il y a maintenant de nouvelles "exigences non fonctionnelles" à considérer également. Par exemple, de nombreux utilisateurs s'opposent à l'utilisation de leurs données privées pour entraîner de nouveaux modèles d'IA. Avez-vous vérifié le niveau de confidentialité fourni par le fournisseur de modèle que vous utilisez ?

Les travailleurs IA comme microservices ?

En lisant à propos de l'approche de la Multitude de Travailleurs, vous pourriez remarquer certaines similitudes avec l'architecture en microservices. Les deux mettent l'accent sur la décomposition de systèmes complexes en unités plus petites, plus gérables et déployables indépendamment. Tout comme les microservices sont conçus pour être faiblement couplés, centrés sur des capacités métier spécifiques et communiquant via des API bien définies, les travailleurs IA sont conçus pour être modulaires, spécialisés dans leurs tâches et interagir entre eux via des interfaces et des protocoles de communication clairs.

Cependant, il y a quelques différences importantes à garder à l'esprit. Alors que les microservices sont généralement implémentés comme des processus ou des services séparés s'exécutant sur différentes machines ou conteneurs, les travailleurs IA peuvent être implémentés comme des composants autonomes au sein d'une seule application ou comme des services séparés, selon vos besoins spécifiques et vos besoins d'évolutivité. De plus, la communication entre les travailleurs IA implique souvent l'échange d'informations riches basées sur le langage naturel, comme des prompts, des instructions et du contenu généré, plutôt que les formats de données plus structurés couramment utilisés dans les microservices.

Malgré ces différences, les principes de modularité, de couplage faible et d'interfaces

de communication claires restent au cœur des deux modèles. En appliquant ces principes à votre architecture de travailleurs IA, vous pouvez créer des systèmes flexibles, évolutifs et maintenables qui exploitent la puissance de l'IA pour résoudre des problèmes complexes et apporter de la valeur à vos utilisateurs.

L'approche de la Multitude de Travailleurs peut être appliquée dans divers domaines et applications, exploitant la puissance de l'IA pour aborder des tâches complexes et fournir des solutions intelligentes. Explorons quelques exemples concrets de la façon dont les travailleurs IA peuvent être employés dans différents contextes.

Gestion des comptes

Pratiquement chaque application web autonome possède le concept de compte (ou utilisateur). Dans Olympia, nous utilisons un travailleur IA AccountManager qui est programmé pour pouvoir gérer différents types de demandes de modification liées aux comptes utilisateurs.

Sa directive se présente comme suit :

```
1 You are an intelligent account manager for Olympia. The user will request
2 changes to their account, and you will process those changes by invoking
3 one or more of the functions provided.
4
5 The initial state of the account : #{account.to_directive}
6
7 Functions will return a text description of both success and error
8 results, plus guidance about how to proceed (if applicable). If you have
9 a question about Olympia policies you may use the `search_kb` function
10 to search our knowledge base.
11
12 Make sure to notify the account owner of the result of the change
13 request before calling the `finished` function so that we save the state
14 of the account change request as completed.
```

L'état initial du compte produit par `account.to_directive` est simplement une description textuelle du compte, incluant les données pertinentes associées telles que les utilisateurs, les abonnements, etc.

La gamme de fonctions disponibles pour l'AccountManager lui permet de modifier l'abonnement de l'utilisateur, d'ajouter et de supprimer des consultants IA et d'autres types de modules complémentaires payants, et d'envoyer des e-mails de notification au propriétaire du compte. En plus de la fonction `finished`, il peut également `notify_human_administrator` s'il rencontre une erreur pendant son traitement ou s'il a besoin d'une assistance quelconque pour une demande.

Notez qu'en cas de questions, l'AccountManager peut choisir de rechercher dans la base de connaissances d'Olympia, où il peut trouver des instructions sur la façon de gérer les cas particuliers et toute autre situation où il n'est pas sûr de la manière de procéder.

Applications E-commerce

Dans le domaine de l'e-commerce, les travailleurs IA peuvent jouer un rôle crucial dans l'amélioration de l'expérience utilisateur et l'optimisation des opérations commerciales. Voici quelques façons dont les travailleurs IA peuvent être utilisés :

Recommandations de Produits

L'une des applications les plus puissantes des travailleurs IA dans l'e-commerce est la génération de recommandations de produits personnalisées. En analysant le comportement des utilisateurs, l'historique des achats et les préférences, ces travailleurs peuvent suggérer des produits adaptés aux intérêts et aux besoins de chaque utilisateur.

La clé pour des recommandations de produits efficaces réside dans l'utilisation combinée des techniques de filtrage collaboratif et de filtrage basé sur le contenu. Le filtrage collaboratif examine le comportement d'utilisateurs similaires pour identifier des modèles et faire des recommandations basées sur ce que d'autres personnes aux goûts similaires ont acheté ou apprécié. Le filtrage basé sur le contenu, quant à lui, se concentre sur les caractéristiques et les attributs des produits eux-mêmes, recommandant des articles qui partagent des caractéristiques similaires à ceux pour lesquels un utilisateur a précédemment montré de l'intérêt.

Voici un exemple simplifié de la façon dont vous pouvez implémenter un travailleur de recommandation de produits en Ruby, cette fois en utilisant un style de programmation fonctionnelle “Railway Oriented (ROP)” :

```
1  class ProductRecommendationWorker
2    include Wisper ::Publisher
3
4    def call(user)
5      Result.ok(ProductRecommendation.new(user))
6        .and_then(ValidateUser.method( :validate))
7        .map>AnalyzeCurrentSession.method( :analyze))
8        .map(CollaborativeFilter.method( :filter))
9        .map(ContentBasedFilter.method( :filter))
10       .map(ProductSelector.method( :select)).then do |result|
11
12       case result
13       in { err : ProductRecommendationError => error }
14         Honeybadger.notify(error.message, context : {user :})
15       in { ok : ProductRecommendations => recs }
16         broadcast( :new_recommendations, user : , recs :)
```

```
17      end
18  end
19 end
20 end
```



Le style de programmation fonctionnelle Ruby utilisé dans l'exemple est influencé par F# et Rust. Vous pouvez en savoir plus à ce sujet dans [l'explication de la technique](#) de mon ami Chad Wooley chez GitLab

Dans cet exemple, le `ProductRecommendationWorker` prend un utilisateur en entrée et génère des recommandations de produits personnalisées en faisant passer un objet de valeur à travers une chaîne d'étapes fonctionnelles. Analysons chaque étape :

1. `ValidateUser.validate` : Cette étape s'assure que l'utilisateur est valide et éligible pour recevoir des recommandations personnalisées. Elle vérifie si l'utilisateur existe, est actif et dispose des données nécessaires pour générer des recommandations. Si la validation échoue, un résultat d'erreur est renvoyé et la chaîne est court-circuitée.
2. `AnalyzeCurrentSession.analyze` : Si l'utilisateur est valide, cette étape analyse sa session de navigation actuelle pour recueillir des informations contextuelles. Elle examine les interactions récentes de l'utilisateur, comme les produits consultés, les requêtes de recherche et le contenu du panier, pour comprendre ses intérêts et intentions actuels.
3. `CollaborativeFilter.filter` : En utilisant le *comportement d'utilisateurs similaires*, cette étape applique des techniques de filtrage collaboratif pour identifier les produits susceptibles d'intéresser l'utilisateur. Elle prend en compte des facteurs comme l'historique d'achats, les évaluations et les interactions utilisateur-produit pour générer un ensemble de recommandations candidates.
4. `ContentBasedFilter.filter` : Cette étape affine davantage les recommandations candidates en appliquant un filtrage basé sur le contenu.

Elle compare les attributs et caractéristiques des produits candidats avec les *préférences et données historiques de l'utilisateur* pour sélectionner les articles les plus pertinents.

5. `ProductSelector.select` : Enfin, cette étape sélectionne les N meilleurs produits parmi les recommandations filtrées selon des critères prédéfinis, tels que le score de pertinence, la popularité ou d'autres règles métier. Les produits sélectionnés sont ensuite renvoyés comme recommandations personnalisées finales.

La beauté d'utiliser un style de programmation fonctionnelle en Ruby ici est qu'il nous permet d'enchaîner ces étapes de manière claire et concise. Chaque étape se concentre sur une tâche spécifique et renvoie un objet `Result`, qui peut être soit un succès (`ok`) soit une erreur (`err`). Si une étape rencontre une erreur, la chaîne est court-circuitée et l'erreur est propagée jusqu'au résultat final.

Dans l'instruction `case` à la fin, nous effectuons un filtrage par motif sur le résultat final. Si le résultat est une erreur (`ProductRecommendationError`), nous journalisons l'erreur en utilisant un outil comme Honeybadger pour le monitoring et le débogage. Si le résultat est un succès (`ProductRecommendations`), nous diffusons un événement `:new_recommendations` en utilisant la bibliothèque pub/sub Wisper, en transmettant l'utilisateur et les recommandations générées.

En tirant parti des techniques de programmation fonctionnelle, nous pouvons créer un worker de recommandation de produits modulaire et maintenable. Chaque étape est autonome et peut être facilement testée, modifiée ou remplacée sans affecter le flux global. L'utilisation du filtrage par motif et de la classe `Result` nous aide à gérer les erreurs avec élégance et garantit que le worker échoue rapidement si une étape rencontre un problème.

Bien sûr, il s'agit d'un exemple simplifié, et dans un scénario réel, vous devriez vous intégrer à votre plateforme e-commerce, gérer les cas limites, et même vous aventurer dans l'implémentation des algorithmes de recommandation. Cependant, les principes

fondamentaux de décomposition du problème en petites étapes et d'utilisation des techniques de programmation fonctionnelle restent les mêmes.

Détection de fraude

Voici un exemple simplifié de la façon dont vous pouvez implémenter un worker de détection de fraude en utilisant le même style de Programmation Orientée Railway (ROP) en Ruby :

```

1  class FraudDetectionWorker
2    include Wisper ::Publisher
3
4    def call(transaction)
5      Result.ok(FraudDetection.new(transaction))
6        .and_then(ValidateTransaction.method( :validate))
7        .map(AnalyzeTransactionPatterns.method( :analyze))
8        .map(CheckCustomerHistory.method( :check))
9        .map(EvaluateRiskFactors.method( :evaluate))
10       .map(DetermineFraudProbability.method( :determine)).then do |result|
11
12       case result
13       in { err : FraudDetectionError => error }
14         Honeybadger.notify(error.message, context : {transaction :})
15       in { ok : FraudDetection => fraud } }
16         if fraud.high_risk ?
17           broadcast( :high_risk_transaction, transaction :, fraud :)
18         else
19           broadcast( :low_risk_transaction, transaction :)
20         end
21       end
22     end
23   end
24 end

```

La classe `FraudDetection` est un *value object* qui encapsule l'état de détection de fraude pour une transaction donnée. Elle fournit une approche structurée pour analyser et évaluer le risque de fraude associé à une transaction en fonction de divers facteurs de risque.

```
1  class FraudDetection
2    RISK_THRESHOLD = 0.8
3
4    attr_accessor :transaction, :risk_factors
5
6    def initialize(transaction)
7      self.transaction = transaction
8      self.risk_factors = []
9    end
10
11   def add_risk_factor(description :, probability :)
12     case { description :, probability : }
13     in { description : String => desc, probability : Float => prob }
14     risk_factors << { desc => prob }
15   else
16     raise ArgumentError, "Risk factor arguments should be string and float"
17   end
18 end
19
20 def high_risk ?
21   fraud_probability > RISK_THRESHOLD
22 end
23
24 private
25
26 def fraud_probability
27   risk_factors.values.sum
28 end
29 end
```

La classe FraudDetection possède les attributs suivants :

- `transaction` : Une référence à la transaction en cours d'analyse pour la fraude.
- `risk_factors` : Un tableau qui stocke les facteurs de risque associés à la transaction. Chaque facteur de risque est représenté sous forme d'une table de hachage, où la clé est la description du facteur de risque, et la valeur est la probabilité de fraude associée à ce facteur de risque.

La méthode `add_risk_factor` permet d'ajouter un facteur de risque au tableau

`risk_factors`. Elle prend deux paramètres : `description`, qui est une chaîne de caractères décrivant le facteur de risque, et `probability`, qui est un nombre à virgule flottante représentant la probabilité de fraude associée à ce facteur de risque. Nous utilisons une condition `case .. in` pour effectuer une simple vérification de type.

La méthode `high_risk?` qui sera vérifiée à la fin de la chaîne est une méthode prédictive qui compare la `fraud_probability` (calculée en additionnant les probabilités de tous les facteurs de risque) avec le `RISK_THRESHOLD`.

La classe `FraudDetection` offre une façon propre et encapsulée de gérer la détection de fraude pour une transaction. Elle permet d'ajouter plusieurs facteurs de risque, chacun avec sa propre description et probabilité, et fournit une méthode pour déterminer si la transaction est considérée à haut risque en fonction de la probabilité de fraude calculée. La classe peut être facilement intégrée dans un système plus large de détection de fraude, où différents composants peuvent collaborer pour évaluer et atténuer le risque de transactions frauduleuses.

Enfin, puisqu'il s'agit après tout d'un livre sur la programmation utilisant l'IA, voici un exemple d'implémentation de la classe `CheckCustomerHistory` exploitant le traitement par IA en utilisant le module `ChatCompletion` de ma bibliothèque [Raix](#) :

```
1  class CheckCustomerHistory
2      include Raix ::ChatCompletion
3
4      attr_accessor :fraud_detection
5
6      INSTRUCTION = <<<~END
7          You are an AI assistant tasked with checking a customer's transaction
8          history for potential fraud indicators. Given the current transaction
9          and the customer's past transactions, analyze the data to identify any
10         suspicious patterns or anomalies.
11
12         Consider factors such as the frequency of transactions, transaction
13         amounts, geographical locations, and any deviations from the customer's
14         typical behavior to generate a probability score as a float in the range
15         of 0 to 1 (with 1 being absolute certainty of fraud).
16
```

```
17     Output the results of your analysis, highlighting any red flags or areas
18     of concern in the following JSON format :
19
20     { description : <Summary of your findings>, probability : <Float> }
21 END
22
23 def self.check(fraud_detection)
24   new(fraud_detection).call
25 end
26
27 def call
28   chat_completion(json : true).tap do |result|
29     fraud_detection.add_risk_factor(**result)
30   end
31   Result.ok(fraud_detection)
32 rescue StandardError => e
33   Result.err(FraudDetectionError.new(e))
34 end
35
36 private
37
38 def initialize(fraud_detection)
39   self.fraud_detection = fraud_detection
40 end
41
42 def transcript
43   tx_history = fraud_detection.transaction.user.tx_history
44   [
45     { system : INSTRUCTION },
46     { user : "Transaction history : #{tx_history.to_json}" },
47     { assistant : "OK. Please provide the current transaction." },
48     { user : "Current transaction : #{fraud_detection.transaction.to_json}" }
49   ]
50 end
51 end
```

Dans cet exemple, la classe `CheckCustomerHistory` définit une constante `INSTRUCTION` qui fournit des instructions spécifiques au modèle d'IA sur la façon d'analyser l'historique des transactions du client pour détecter des indicateurs potentiels de fraude via une directive système

La méthode `self.check` est une méthode de classe qui initialise une nouvelle instance de `CheckCustomerHistory` avec l'objet `fraud_detection` et appelle la méthode `call` pour effectuer l'analyse de l'historique du client.

Dans la méthode `call`, l'historique des transactions du client est récupéré et formaté en un transcript qui est transmis au modèle d'IA. Le modèle d'IA analyse l'historique des transactions selon les instructions fournies et renvoie un résumé de ses conclusions.

Les conclusions sont ajoutées à l'objet `fraud_detection`, et l'objet `fraud_detection` mis à jour est renvoyé comme `Result` réussi.

En s'appuyant sur le module `ChatCompletion`, la classe `CheckCustomerHistory` peut utiliser la puissance de l'IA pour analyser l'historique des transactions du client et identifier les indicateurs potentiels de fraude. Cela permet des techniques de détection de fraude plus sophistiquées et adaptatives, car le modèle d'IA peut apprendre et s'adapter à de nouveaux modèles et anomalies au fil du temps.

Le `FraudDetectionWorker` mis à jour et la classe `CheckCustomerHistory` démontrent comment les workers IA peuvent être intégrés de manière transparente, améliorant le processus de détection de fraude avec des capacités d'analyse et de prise de décision intelligentes.

Analyse du Sentiment Client

Voici un autre exemple similaire de la façon dont vous pouvez implémenter un worker d'analyse du sentiment client. Beaucoup moins d'explications cette fois-ci, puisque vous devriez comprendre le principe de ce style de programmation :

```
1  class CustomerSentimentAnalysisWorker
2    include Wisper ::Publisher
3
4    def call(feedback)
5      Result.ok(feedback)
6        .and_then(PreprocessFeedback.method( :preprocess))
7        .map(PerformSentimentAnalysis.method( :analyze))
8        .map(ExtractKeyPhrases.method( :extract))
9        .map(IdentifyTrends.method( :identify))
10       .map(GenerateInsights.method( :generate)).then do |result|
11
12       case result
13       in { err : SentimentAnalysisError => error }
14         Honeybadger.notify(error.message, context : {feedback : })
15       in { ok : SentimentAnalysisResult => result }
16         broadcast( :sentiment_analysis_completed, result)
17       end
18     end
19   end
20 end
```

Dans cet exemple, le `CustomerSentimentAnalysisWorker` comprend des étapes telles que le prétraitement des commentaires (par exemple, l'élimination du bruit, la tokenisation), l'analyse des sentiments pour déterminer le sentiment global (positif, négatif ou neutre), l'extraction des phrases clés et des sujets, l'identification des tendances et des modèles, et la génération d'insights exploitables basés sur l'analyse.

Applications dans le domaine de la santé

Dans le domaine de la santé, les travailleurs IA peuvent assister les professionnels de santé et les chercheurs dans diverses tâches, conduisant à l'amélioration des résultats pour les patients et à l'accélération des découvertes médicales. Voici quelques exemples :

Admission des patients

Les travailleurs IA peuvent rationaliser le processus d'admission des patients en automatisant diverses tâches et en fournissant une assistance intelligente.

Planification des rendez-vous : Les travailleurs IA peuvent gérer la planification des rendez-vous en comprenant les préférences des patients, leur disponibilité et l'urgence de leurs besoins médicaux. Ils peuvent interagir avec les patients via des interfaces conversationnelles, les guidant à travers le processus de planification et trouvant les créneaux de rendez-vous les plus appropriés en fonction des exigences du patient et de la disponibilité du prestataire de soins.

Collecte des antécédents médicaux : Lors de l'admission des patients, les travailleurs IA peuvent aider à collecter et à documenter les antécédents médicaux du patient. Ils peuvent engager des dialogues interactifs avec les patients, posant des questions pertinentes sur leurs antécédents médicaux, leurs médicaments, leurs allergies et leurs antécédents familiaux. Les travailleurs IA peuvent utiliser des techniques de traitement du langage naturel pour interpréter et structurer les informations collectées, garantissant qu'elles sont correctement enregistrées dans le dossier médical électronique du patient.

Évaluation et stratification des symptômes : Les travailleurs IA peuvent effectuer des évaluations initiales des symptômes en interrogeant les patients sur leurs symptômes actuels, leur durée, leur gravité et tout facteur associé. En s'appuyant sur des bases de connaissances médicales et des modèles d'apprentissage automatique, ces travailleurs peuvent analyser les informations fournies et générer des diagnostics différentiels préliminaires ou recommander les prochaines étapes appropriées, comme la programmation d'une consultation avec un prestataire de soins ou la suggestion de mesures d'auto-soins.

Vérification d'assurance : Les travailleurs IA peuvent aider à la vérification d'assurance lors de l'admission des patients. Ils peuvent collecter les détails d'assurance du patient, communiquer avec les assureurs via des API ou des services web, et vérifier l'éligibilité

de la couverture et les prestations. Cette automatisation aide à rationaliser le processus de vérification d'assurance, réduisant la charge administrative et assurant une capture précise des informations.

Éducation et instructions aux patients : Les travailleurs IA peuvent fournir aux patients du matériel éducatif et des instructions pertinentes basés sur leurs conditions médicales spécifiques ou leurs procédures à venir. Ils peuvent délivrer du contenu personnalisé, répondre aux questions courantes et offrir des conseils sur les préparations pré-rendez-vous, les instructions concernant les médicaments ou les soins post-traitement. Cela aide à maintenir les patients informés et engagés tout au long de leur parcours de soins.

En utilisant des travailleurs IA dans l'admission des patients, les organisations de santé peuvent améliorer l'efficacité, réduire les temps d'attente et améliorer l'expérience globale du patient. Ces travailleurs peuvent gérer les tâches routinières, collecter des informations précises et fournir une assistance personnalisée, permettant aux professionnels de santé de se concentrer sur la prestation de soins de haute qualité aux patients.

Évaluation des risques des patients

Les travailleurs IA peuvent jouer un rôle crucial dans l'évaluation des risques des patients en analysant diverses sources de données et en appliquant des techniques d'analyse avancées.

Intégration des données : Les travailleurs IA peuvent rassembler et donner un sens aux données des patients provenant de multiples sources, telles que les dossiers médicaux électroniques (DME), l'imagerie médicale, les résultats de laboratoire, les appareils portables et les déterminants sociaux de la santé. En consolidant ces informations en un profil patient complet, les travailleurs IA peuvent fournir une vue holistique de l'état de santé du patient et de ses facteurs de risque.

Stratification des risques : Les travailleurs IA peuvent utiliser des modèles prédictifs pour stratifier les patients en différentes catégories de risque basées sur leurs caractéristiques individuelles et leurs données de santé. Cette stratification des risques permet aux prestataires de soins de santé de prioriser les patients qui nécessitent une attention ou une intervention plus immédiate. Par exemple, les patients identifiés comme étant à haut risque pour une condition particulière peuvent être signalés pour un suivi plus étroit, des mesures préventives ou une intervention précoce.

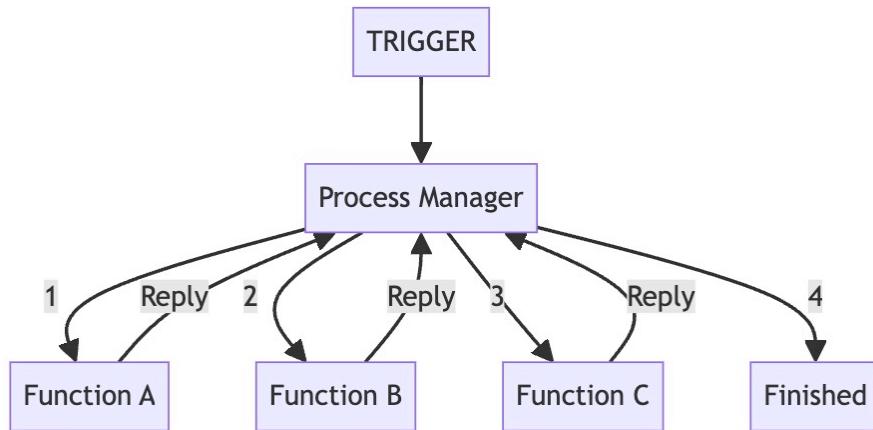
Profils de risque personnalisés : Les travailleurs IA peuvent générer des profils de risque personnalisés pour chaque patient, mettant en évidence les facteurs spécifiques contribuant à leurs scores de risque. Ces profils peuvent inclure des aperçus du mode de vie du patient, des prédispositions génétiques, des facteurs environnementaux et des déterminants sociaux de la santé. En fournissant une analyse détaillée des facteurs de risque, les travailleurs IA peuvent aider les prestataires de soins de santé à adapter les stratégies de prévention et les plans de traitement aux besoins individuels des patients.

Surveillance continue des risques : Les travailleurs IA peuvent surveiller en continu les données des patients et mettre à jour les évaluations des risques en temps réel. Lorsque de nouvelles informations deviennent disponibles, comme des changements dans les signes vitaux, les résultats de laboratoire ou l'adhésion aux médicaments, les travailleurs IA peuvent recalculer les scores de risque et alerter les prestataires de soins de santé de tout changement significatif. Cette surveillance proactive permet des interventions opportunes et des ajustements aux plans de soins des patients.

Aide à la décision clinique : Les travailleurs IA peuvent intégrer les résultats de l'évaluation des risques dans les systèmes d'aide à la décision clinique, fournissant aux prestataires de soins de santé des recommandations et des alertes basées sur des preuves. Par exemple, si le score de risque d'un patient pour une condition particulière dépasse un certain seuil, le travailleur IA peut inciter le prestataire de soins à envisager des tests diagnostiques spécifiques, des mesures préventives ou des options de traitement basées sur les directives cliniques et les meilleures pratiques.

Ces agents peuvent traiter de vastes quantités de données patients, appliquer des analyses sophistiquées et générer des informations exploitables pour soutenir la prise de décision clinique. Cela conduit finalement à l'amélioration des résultats pour les patients, à la réduction des coûts de santé et à une meilleure gestion de la santé des populations.

L'Agent IA comme Gestionnaire de Processus



Dans le contexte des applications pilotées par l'IA, un agent peut être conçu pour fonctionner comme un Gestionnaire de Processus, tel que décrit dans le livre “Enterprise Integration Patterns” par Gregor Hohpe. Un Gestionnaire de Processus est un composant central qui maintient l'état d'un processus et détermine les prochaines étapes de traitement en fonction des résultats intermédiaires.

Lorsqu'un agent IA agit en tant que Gestionnaire de Processus, il reçoit un message entrant qui initialise le processus, appelé *message déclencheur*. L'agent IA maintient ensuite l'état de l'exécution du processus (sous forme de transcription de conversation)

et traite le message à travers une série d'étapes de traitement implémentées comme des fonctions d'outil, qui peuvent être séquentielles ou parallèles, et appelées à sa discrédition.



Si vous utilisez une classe de modèle d'IA comme GPT-4 qui sait comment exécuter des fonctions en parallèle, votre agent peut alors exécuter plusieurs étapes simultanément. J'admet que je n'ai pas essayé de le faire moi-même et mon instinct me dit que les résultats peuvent varier.

Après chaque étape de traitement individuelle, le contrôle est rendu à l'agent IA, lui permettant de déterminer la ou les prochaines étapes de traitement en fonction de l'état actuel et des résultats obtenus.

Stockez Vos Messages Déclencheurs

D'après mon expérience, il est judicieux d'implémenter votre message déclencheur comme un objet supporté par une base de données. De cette façon, chaque instance de processus est identifiée par une clé primaire unique et vous donne un endroit pour stocker l'état associé à l'exécution, y compris la transcription de conversation de l'IA.

Par exemple, voici une version simplifiée de la classe modèle `AccountChange` d'Olympia, qui représente une demande de modification du compte d'un utilisateur.

```
1  # == Schema Information
2  #
3  # Table name : account_changes
4  #
5  # id          :uuid          not null, primary key
6  # description :string
7  # state       :string        not null
8  # transcript  :jsonb
9  # created_at  :datetime      not null
10 # updated_at  :datetime      not null
11 # account_id  :uuid          not null
12 #
13 # Indexes
14 #
15 #   index_account_changes_on_account_id  (account_id)
16 #
17 # Foreign Keys
18 #
19 #   fk_rails_...  (account_id => accounts.id)
20 #
21 class AccountChange < ApplicationRecord
22   belongs_to  :account
23
24   validates  :description, presence : true
25
26   after_commit -> {
27     broadcast( :account_change_requested, self)
28   }, on :  :create
29
30   state_machine initial : :requested do
31     event  :completed do
32       transition all =>  :complete
33     end
34     event  :failed do
35       transition all =>  :requires_human_review
36     end
37   end
38 end
```

La classe AccountChange sert de message déclencheur qui initie un processus pour gérer la demande de modification de compte. Notez comment elle est diffusée vers

le sous-système de publication-abonnement basé sur [Wisper](#) d’Olympia après que la transaction de création ait terminé sa validation.

Le stockage du message déclencheur dans la base de données de cette manière fournit un enregistrement permanent de chaque demande de modification de compte. Chaque instance de la classe `AccountChange` se voit attribuer une clé primaire unique, permettant une identification et un suivi faciles des demandes individuelles. Ceci est particulièrement utile pour la journalisation d’audit, car cela permet au système de maintenir un historique de toutes les modifications de compte, y compris quand elles ont été demandées, quelles modifications ont été demandées et l’état actuel de chaque demande.

Dans l’exemple donné, la classe `AccountChange` inclut des champs tels que `description` pour capturer les détails de la modification demandée, `state` pour représenter l’état actuel de la demande (par exemple, demandée, terminée, nécessite_révision_humaine), et `transcript` pour stocker la transcription de la conversation avec l’IA liée à la demande. Le champ `description` est la requête réelle utilisée pour initier la première completion de chat avec l’IA. Le stockage de ces données fournit un contexte précieux et permet un meilleur suivi et une meilleure analyse du processus de modification de compte.

Le stockage des messages déclencheurs dans la base de données permet une gestion robuste des erreurs et de la récupération. Si une erreur survient pendant le traitement d’une demande de modification de compte, le système marque la demande comme échouée et la fait passer à un état qui nécessite une intervention humaine. Cela garantit qu’aucune demande n’est perdue ou oubliée, et que tout problème peut être correctement traité et résolu.

Le travailleur IA, en tant que Gestionnaire de processus, fournit un point de contrôle central et permet de puissantes capacités de rapport et de débogage des processus.

Cependant, il est important de noter que l'utilisation d'un travailleur IA comme Gestionnaire de processus pour chaque scénario de flux de travail dans votre application peut être excessif.

Intégration des travailleurs IA dans l'architecture de votre application

Lors de l'incorporation de travailleurs IA dans l'architecture de votre application, plusieurs considérations techniques doivent être abordées pour assurer une intégration fluide et une communication efficace entre les travailleurs IA et les autres composants de l'application. Cette section examine les aspects clés de la conception de ces interfaces, de la gestion du flux de données et de la gestion du cycle de vie des travailleurs IA.

Conception d'interfaces claires et de protocoles de communication

Pour faciliter une intégration transparente entre les travailleurs IA et les autres composants de l'application, il est crucial de définir des interfaces claires et des protocoles de communication. Considérez les approches suivantes :

Intégration basée sur API : Exposez la fonctionnalité des travailleurs IA à travers des API bien définies, comme des points d'accès RESTful ou des schémas GraphQL. Cela permet aux autres composants d'interagir avec les travailleurs IA en utilisant des requêtes et réponses HTTP standard. L'intégration basée sur API fournit un contrat clair entre les travailleurs IA et les composants consommateurs, facilitant le développement, les tests et la maintenance des points d'intégration.

Communication basée sur les messages : Implémentez des modèles de communication basés sur les messages, tels que les files d'attente de messages ou les systèmes de publication-abonnement, pour permettre une interaction asynchrone entre les

travailleurs IA et les autres composants. Cette approche découpe les travailleurs IA du reste de l'application, permettant une meilleure évolutivité, tolérance aux pannes et un couplage faible. La communication basée sur les messages est particulièrement utile lorsque le traitement effectué par les travailleurs IA est long ou gourmand en ressources, car elle permet aux autres parties de l'application de continuer à s'exécuter sans attendre que les travailleurs IA terminent leurs tâches.

Architecture événementielle : Concevez votre système autour d'événements et de déclencheurs qui activent les travailleurs IA lorsque des conditions spécifiques sont remplies. Les travailleurs IA peuvent s'abonner aux événements pertinents et réagir en conséquence, effectuant leurs tâches désignées lorsque les événements se produisent. L'architecture événementielle permet un traitement en temps réel et permet aux travailleurs IA d'être invoqués à la demande, réduisant la consommation inutile de ressources. Cette approche est bien adaptée aux scénarios où les travailleurs IA doivent répondre à des actions spécifiques ou à des changements dans l'état de l'application.

Gestion du flux de données et synchronisation

Lors de l'intégration de travailleurs IA dans votre application, il est crucial d'assurer un flux de données fluide et de maintenir la cohérence des données entre les travailleurs IA et les autres composants. Considérez les aspects suivants :

Préparation des données : Avant d'alimenter les travailleurs IA en données, vous devrez peut-être effectuer diverses tâches de préparation des données, comme le nettoyage, le formatage et/ou la transformation des données d'entrée. Vous voulez non seulement vous assurer que les travailleurs IA peuvent traiter efficacement, mais aussi vous assurer que vous ne gaspillez pas de jetons en accordant de l'attention à des informations que le travailleur pourrait considérer comme inutiles au mieux, et distrayantes au pire. La préparation des données peut impliquer des tâches comme la suppression du bruit, la gestion des valeurs manquantes ou la conversion des types de données.

Persistance des données : Comment allez-vous stocker et persister les données qui circulent dans et hors des travailleurs IA ? Considérez des facteurs comme le volume de données, les modèles de requêtes et l'évolutivité. Avez-vous besoin de persister la transcription de l'IA comme reflet de son “processus de réflexion” à des fins d'audit ou de débogage, ou est-il suffisant d'avoir uniquement un enregistrement des résultats ?

Récupération des données : L'obtention des données nécessaires aux workers peut impliquer l'interrogation de bases de données, la lecture de fichiers ou l'accès à des API externes. Assurez-vous de prendre en compte la latence et la manière dont les workers IA auront accès aux données les plus récentes. Ont-ils besoin d'un accès complet à votre base de données ou devriez-vous définir étroitement la portée de leur accès en fonction de leurs tâches ? Qu'en est-il de la mise à l'échelle ? Considérez les mécanismes de mise en cache pour améliorer les performances et réduire la charge sur les sources de données sous-jacentes.

Synchronisation des données : Lorsque plusieurs composants, y compris les workers IA, accèdent et modifient des données partagées, il est important de mettre en œuvre des mécanismes de synchronisation appropriés pour maintenir la cohérence des données. Les stratégies de verrouillage des bases de données, telles que le verrouillage optimiste ou pessimiste, peuvent vous aider à prévenir les conflits et à garantir l'intégrité des données. Mettez en œuvre des techniques de gestion des transactions pour regrouper les opérations de données connexes et maintenir les propriétés ACID (atomicité, cohérence, isolation et durabilité)

Gestion des erreurs et récupération : Mettez en œuvre des mécanismes robustes de gestion des erreurs et de récupération pour traiter les problèmes liés aux données qui peuvent survenir pendant le processus de flux de données. Gérez les exceptions avec élégance et fournissez des messages d'erreur significatifs pour faciliter le débogage. Mettez en œuvre des mécanismes de nouvelle tentative et des stratégies de repli pour gérer les défaillances temporaires ou les interruptions réseau. Définissez des procédures claires pour la récupération et la restauration des données en cas de corruption ou de

perte.

En concevant et en mettant en œuvre soigneusement les mécanismes de flux et de synchronisation des données, vous pouvez garantir que vos workers IA ont accès à des données précises, cohérentes et à jour. Cela leur permet d'effectuer leurs tâches efficacement et de produire des résultats fiables.

Gestion du cycle de vie des workers IA

Développez un processus standardisé pour l'initialisation et la configuration des workers IA. Je privilégie les frameworks qui standardisent la façon de définir les paramètres tels que les noms de modèles, les directives système et les définitions de fonctions. Assurez-vous que le processus d'initialisation est automatisé et reproductible pour faciliter le déploiement et la mise à l'échelle.

Mettez en œuvre des mécanismes complets de surveillance et de journalisation pour suivre la santé et les performances des workers IA. Collectez des métriques telles que l'utilisation des ressources, le temps de traitement, les taux d'erreur et le débit. Utilisez des systèmes de journalisation centralisés comme la pile ELK (Elasticsearch, Logstash, Kibana) pour agréger et analyser les journaux de plusieurs workers IA.

Intégrer la tolérance aux pannes et la résilience dans l'architecture des workers IA. Mettez en œuvre des mécanismes de gestion des erreurs et de récupération pour gérer élégamment les défaillances ou les exceptions. Les Grands Modèles de Langage sont encore une technologie de pointe; les fournisseurs ont tendance à tomber en panne souvent à des moments inattendus. Utilisez des mécanismes de nouvelle tentative et des disjoncteurs pour éviter les défaillances en cascade.

Composabilité et orchestration des workers IA

L'un des principaux avantages de l'architecture des workers IA est sa composabilité, qui permet de combiner et d'orchestrer plusieurs workers IA pour résoudre des problèmes

complexes. En décomposant une tâche plus importante en sous-tâches plus petites et plus gérables, chacune traitée par un worker IA spécialisé, vous pouvez créer des systèmes puissants et flexibles. Dans cette section, nous explorerons différentes approches pour composer et orchestrer “une multitude” de workers IA.

Chaînage des workers IA pour des flux de travail multi-étapes

Dans de nombreux scénarios, une tâche complexe peut être décomposée en une série d'étapes séquentielles, où la sortie d'un worker IA devient l'entrée du suivant. Ce chaînage de workers IA crée un flux de travail ou un pipeline multi-étapes. Chaque worker IA dans la chaîne se concentre sur une sous-tâche spécifique, et la sortie finale est le résultat des efforts combinés de tous les workers.

Prenons un exemple dans le contexte d'une application Ruby on Rails pour le traitement de contenu généré par l'utilisateur. Le flux de travail implique les étapes suivantes, qui sont probablement chacune trop simples pour justifier une décomposition de cette manière dans des cas d'utilisation réels, mais elles rendent l'exemple plus facile à comprendre :

- 1. Nettoyage du texte** : Un worker IA responsable de la suppression des balises HTML, de la conversion du texte en minuscules et de la gestion de la normalisation Unicode.
- 2. Détection de la langue** : Un worker IA qui identifie la langue du texte nettoyé.
- 3. Analyse des sentiments** : Un worker IA qui détermine le sentiment (positif, négatif ou neutre) du texte en fonction de la langue détectée.
- 4. Catégorisation du contenu** : Un worker IA qui classe le texte dans des catégories prédéfinies en utilisant des techniques de traitement du langage naturel.

Voici un exemple très simplifié de la façon dont vous pouvez chaîner ces workers IA ensemble en utilisant Ruby :

```
1  class ContentProcessor
2    def initialize(text)
3      @text = text
4    end
5
6    def process
7      cleaned_text = TextCleanupWorker.new(@text).call
8      language = LanguageDetectionWorker.new(cleaned_text).call
9      sentiment = SentimentAnalysisWorker.new(cleaned_text, language).call
10     category = CategorizationWorker.new(cleaned_text, language).call
11
12     { cleaned_text : , language : , sentiment : , category : }
13   end
14 end
```

Dans cet exemple, la classe `ContentProcessor` s'initialise avec le texte brut et enchaîne les agents d'IA dans la méthode `process`. Chaque agent d'IA effectue sa tâche spécifique et transmet le résultat à l'agent suivant dans la chaîne. La sortie finale est un hash contenant le texte nettoyé, la langue détectée, le sentiment et la catégorie du contenu.

Traitement parallèle pour les agents d'IA indépendants

Dans l'exemple précédent, les agents d'IA sont chainés de manière séquentielle, où chaque agent traite le texte et transmet le résultat à l'agent suivant. Cependant, si vous disposez de plusieurs agents d'IA pouvant fonctionner indépendamment sur la même entrée, vous pouvez optimiser le flux de travail en les traitant en parallèle.

Dans le scénario donné, une fois que le nettoyage du texte est effectué par le `TextCleanupWorker`, le `LanguageDetectionWorker`, `SentimentAnalysisWorker`, et `CategorizationWorker` peuvent tous traiter le texte nettoyé de manière indépendante. En exécutant ces agents en parallèle, vous pouvez potentiellement réduire le temps de traitement global et améliorer l'efficacité de votre flux de travail.

Pour réaliser le traitement parallèle en Ruby, vous pouvez utiliser des techniques de concurrence telles que les threads ou la programmation asynchrone. Voici un exemple

de la façon dont vous pouvez modifier la classe ContentProcessor pour traiter les trois derniers agents en parallèle en utilisant des threads :

```
1  require 'concurrent'
2
3  class ContentProcessor
4    def initialize(text)
5      @text = text
6    end
7
8    def process
9      cleaned_text = TextCleanupWorker.new(@text).call
10
11    language_future = Concurrent::Future.execute do
12      LanguageDetectionWorker.new(cleaned_text).call
13    end
14
15    sentiment_future = Concurrent::Future.execute do
16      SentimentAnalysisWorker.new(cleaned_text).call
17    end
18
19    category_future = Concurrent::Future.execute do
20      CategorizationWorker.new(cleaned_text).call
21    end
22
23    language = language_future.value
24    sentiment = sentiment_future.value
25    category = category_future.value
26
27    { cleaned_text :, language :, sentiment :, category : }
28  end
29 end
```

Dans cette version optimisée, nous utilisons la bibliothèque concurrent-ruby pour créer des objets `Concurrent::Future` pour chacun des workers IA indépendants. Un `Future` représente un calcul qui sera effectué de manière asynchrone dans un thread séparé.

Après l'étape de nettoyage du texte, nous créons trois objets `Future` : `language_future`, `sentiment_future`, et `category_future`. Chaque

Future exécute son worker IA correspondant (`LanguageDetectionWorker`, `SentimentAnalysisWorker`, et `CategorizationWorker`) dans un thread séparé, en passant le `cleaned_text` comme entrée.

En appelant la méthode `value` sur chaque Future, nous attendons que le calcul soit terminé et récupérons le résultat. La méthode `value` bloque jusqu'à ce que le résultat soit disponible, garantissant que tous les workers parallèles ont terminé leur traitement avant de continuer.

Enfin, nous construisons le hash de sortie avec le texte nettoyé et les résultats des workers parallèles, comme dans l'exemple original.

En traitant les workers IA indépendants en parallèle, vous pouvez potentiellement réduire le temps de traitement global par rapport à une exécution séquentielle. Cette optimisation est particulièrement bénéfique lors du traitement de tâches chronophages ou de grands volumes de données.

Cependant, il est important de noter que les gains de performance réels dépendent de divers facteurs, tels que la complexité de chaque worker, les ressources système disponibles et la surcharge liée à la gestion des threads. Il est toujours recommandé de faire des tests de performance et de profiler votre code pour déterminer le niveau optimal de parallélisme pour votre cas d'utilisation spécifique.

De plus, lors de l'implémentation du traitement parallèle, il faut être attentif aux ressources partagées ou aux dépendances entre les workers. Assurez-vous que les workers peuvent fonctionner indépendamment sans conflits ni conditions de concurrence. S'il existe des dépendances ou des ressources partagées, vous devrez peut-être mettre en place des mécanismes de synchronisation appropriés pour maintenir l'intégrité des données et éviter des problèmes tels que les interblocages ou les résultats incohérents.

Le Global Interpreter Lock (GIL) de Ruby et le traitement asynchrone

Il est important de comprendre les implications du Global Interpreter Lock (GIL) de Ruby lorsqu'on envisage un traitement asynchrone basé sur les threads en Ruby.

Le GIL est un mécanisme dans l'interpréteur Ruby qui garantit qu'un seul thread peut exécuter du code Ruby à la fois, même sur des processeurs multi-cœurs. Cela signifie que même si plusieurs threads peuvent être créés et gérés au sein d'un processus Ruby, un seul thread peut activement exécuter du code Ruby à un moment donné.

Le GIL est conçu pour simplifier l'implémentation de l'interpréteur Ruby et assurer la sécurité des threads pour les structures de données internes de Ruby. Cependant, il limite également le potentiel d'exécution véritablement parallèle du code Ruby.

Lorsque vous utilisez des threads en Ruby, comme avec la bibliothèque concurrent-ruby ou la classe Thread intégrée, les threads sont soumis aux contraintes du GIL. Le GIL permet à chaque thread d'exécuter du code Ruby pendant une courte tranche de temps avant de passer à un autre thread, créant l'illusion d'une exécution concurrente.

Cependant, en raison du GIL, l'exécution réelle du code Ruby reste séquentielle. Pendant qu'un thread exécute du code Ruby, les autres threads sont essentiellement en pause, attendant leur tour pour acquérir le GIL et s'exécuter.

Cela signifie que le traitement asynchrone basé sur les threads en Ruby est plus efficace pour les tâches limitées par les E/S, comme l'attente de réponses d'API externes (comme les modèles de langage hébergés par des tiers) ou l'exécution d'opérations d'E/S fichier. Lorsqu'un thread rencontre une opération d'E/S, il peut libérer le GIL, permettant à d'autres threads de s'exécuter pendant l'attente de la fin de l'E/S.

En revanche, pour les tâches limitées par le CPU, comme les calculs intensifs ou

le traitement de workers IA de longue durée, le GIL peut limiter les gains de performance potentiels du parallélisme basé sur les threads. Puisqu'un seul thread peut exécuter du code Ruby à la fois, le temps d'exécution global peut ne pas être significativement réduit par rapport au traitement séquentiel.

Pour obtenir une véritable exécution parallèle pour les tâches limitées par le CPU en Ruby, vous devrez peut-être explorer d'autres approches, telles que :

- Utiliser le parallélisme basé sur les processus avec plusieurs processus Ruby, chacun s'exécutant sur un cœur CPU distinct.
- Exploiter des bibliothèques externes ou des frameworks qui fournissent des extensions natives ou des interfaces vers des langages sans GIL, comme C ou Rust.,
- Utiliser des frameworks de calcul distribué ou des files d'attente de messages pour distribuer les tâches entre plusieurs machines ou processus.

Il est crucial de prendre en compte la nature de vos tâches et les limitations imposées par le GIL lors de la conception et de l'implémentation du traitement asynchrone en Ruby. Bien que le traitement asynchrone basé sur les threads puisse apporter des avantages pour les tâches limitées par les E/S, il peut ne pas offrir d'améliorations significatives des performances pour les tâches limitées par le CPU en raison des contraintes du GIL.

Techniques d'ensemble pour une meilleure précision

Les techniques d'ensemble impliquent la combinaison des sorties de plusieurs workers IA pour améliorer la précision globale ou la robustesse du système. Au lieu de s'appuyer sur un seul worker IA, les techniques d'ensemble exploitent l'intelligence collective de plusieurs workers pour prendre des décisions plus éclairées.



Les ensembles sont particulièrement importants lorsque différentes parties de votre flux de travail fonctionnent mieux avec différents modèles d'IA, une situation plus courante qu'on ne le pense. Les modèles puissants comme GPT-4 sont extrêmement coûteux comparés aux options open source moins performantes, et ne sont probablement pas nécessaires pour chaque étape du flux de travail de votre application.

Une technique d'ensemble courante est le vote majoritaire, où plusieurs agents d'IA traitent indépendamment la même entrée, et la sortie finale est déterminée par le consensus majoritaire. Cette approche peut aider à atténuer l'impact des erreurs individuelles des agents et améliorer la fiabilité globale du système.

Prenons un exemple où nous avons trois agents d'IA pour l'analyse des sentiments, chacun utilisant un modèle différent ou disposant de contextes différents. Nous pouvons combiner leurs sorties en utilisant le vote majoritaire pour déterminer la prédiction finale du sentiment.

```
1  class SentimentAnalysisEnsemble
2      def initialize(text)
3          @text = text
4      end
5
6      def analyze
7          predictions = [
8              SentimentAnalysisWorker1.new(@text).analyze,
9              SentimentAnalysisWorker2.new(@text).analyze,
10             SentimentAnalysisWorker3.new(@text).analyze
11         ]
12
13         predictions
14             .group_by { |sentiment| sentiment }
15             .max_by { |_, votes| votes.size }
16             .first
17
18     end
19 end
```

Dans cet exemple, la classe `SentimentAnalysisEnsemble` s'initialise avec le texte et fait appel à trois différents agents d'IA pour l'analyse des sentiments. La méthode `analyze` recueille les prédictions de chaque agent et détermine le sentiment majoritaire en utilisant les méthodes `group_by` et `max_by`. Le résultat final est le sentiment qui reçoit le plus de votes de l'ensemble des agents.



Les ensembles sont clairement un cas où l'expérimentation avec le parallélisme peut valoir votre temps.

Sélection et Invocation Dynamiques des Agents d'IA

Dans certains cas, sinon la plupart, l'agent d'IA spécifique à invoquer peut dépendre des conditions d'exécution ou des entrées utilisateur. La sélection et l'invocation dynamiques des agents d'IA permettent une flexibilité et une adaptabilité dans le système.



Vous pourriez être tenté d'essayer d'intégrer beaucoup de fonctionnalités dans un seul agent d'IA, en lui donnant de nombreuses fonctions et une invite complexe qui explique comment les appeler. Résistez à cette tentation, faites-moi confiance. L'une des raisons pour lesquelles l'approche dont nous discutons dans ce chapitre s'appelle "Multitude de Travailleurs" est de nous rappeler qu'il est souhaitable d'avoir de nombreux agents spécialisés, chacun accomplissant sa petite tâche au service d'un objectif plus grand.

Par exemple, considérons une application d'agent conversationnel où différents agents d'IA sont responsables du traitement de différents types de requêtes utilisateur. En fonction de l'entrée de l'utilisateur, l'application sélectionne dynamiquement l'agent d'IA approprié pour traiter la requête.

```
1  class ChatbotController < ApplicationController
2    def process_query
3      query = params[ :query]
4      query_type = QueryClassifierWorker.new(query).classify
5
6      case query_type
7      when 'greeting'
8        response = GreetingWorker.new(query).generate_response
9      when 'product_inquiry'
10        response = ProductInquiryWorker.new(query).generate_response
11      when 'order_status'
12        response = OrderStatusWorker.new(query).generate_response
13      else
14        response = DefaultResponseWorker.new(query).generate_response
15      end
16
17      render json : { response : response }
18    end
19  end
```

Dans cet exemple, le ChatbotController reçoit une requête utilisateur via l'action `process_query`. Il utilise d'abord un `QueryClassifierWorker` pour déterminer le type de la requête. En fonction du type de requête classifié, le contrôleur sélectionne dynamiquement l'agent AI approprié pour générer la réponse. Cette sélection dynamique permet au chatbot de gérer différents types de requêtes et de les diriger vers les agents AI pertinents.



Étant donné que le travail du `QueryClassifierWorker` est relativement simple et ne nécessite pas beaucoup de contexte ou de définitions de fonctions, vous pouvez probablement l'implémenter en utilisant un petit LLM ultra-rapide comme [mistralai/mistral-8x7b-instruct:nitro](https://mistralai.github.io/mistral-8x7b-instruct:nitro). Il possède des capacités qui s'approchent du niveau de GPT-4 sur de nombreuses tâches et, au moment où j'écris ces lignes, Groq peut le servir à une vitesse fulgurante de 444 tokens par seconde.

Combiner le TLN traditionnel avec les LLM

Bien que les Grands Modèles de Langage (LLM) aient révolutionné le domaine du traitement du langage naturel (TLN), offrant une polyvalence et des performances inégalées dans un large éventail de tâches, ils ne sont pas toujours la solution la plus efficace ou la plus rentable pour chaque problème. Dans de nombreux cas, la combinaison des techniques traditionnelles de TLN avec les LLM peut conduire à des approches plus optimisées, ciblées et économiques pour résoudre des défis spécifiques de TLN.

Considérez les LLM comme les couteaux suisses du TLN — incroyablement polyvalents et puissants, mais pas nécessairement le meilleur outil pour chaque tâche. Parfois, un outil dédié comme un tire-bouchon ou un ouvre-boîte peut être plus efficace pour une tâche spécifique. De même, les techniques traditionnelles de TLN, comme le regroupement de documents, l'identification des sujets et la classification, peuvent souvent fournir des solutions plus ciblées et rentables pour certains aspects de votre pipeline TLN.

L'un des principaux avantages des techniques traditionnelles de TLN est leur efficacité computationnelle. Ces méthodes, qui reposent souvent sur des modèles statistiques plus simples ou des approches basées sur des règles, peuvent traiter de grands volumes de données textuelles beaucoup plus rapidement et avec moins de ressources computationnelles par rapport aux LLM. Cela les rend particulièrement adaptées aux tâches impliquant l'analyse et l'organisation de grands corpus de documents, comme le regroupement d'articles similaires ou l'identification des sujets clés dans une collection de textes.

De plus, les techniques traditionnelles de TLN peuvent souvent atteindre une haute précision pour des tâches spécifiques, particulièrement lorsqu'elles sont entraînées sur des ensembles de données spécifiques à un domaine. Par exemple, un classificateur de documents bien ajusté utilisant des algorithmes d'apprentissage automatique

traditionnels comme les Machines à vecteurs de support (SVM) ou Naive Bayes peut catégoriser précisément les documents dans des catégories prédéfinies avec un coût computationnel minimal.

Cependant, les LLM excellent vraiment dans les tâches qui nécessitent une compréhension plus profonde du langage, du contexte et du raisonnement. Leur capacité à générer du texte cohérent et contextuellement pertinent, à répondre à des questions et à résumer de longs passages est inégalée par les méthodes traditionnelles de TLN. Les LLM peuvent efficacement gérer des phénomènes linguistiques complexes, comme l'ambiguïté, la coréférence et les expressions idiomatiques, les rendant inestimables pour les tâches qui nécessitent la génération ou la compréhension du langage naturel.

La véritable puissance réside dans la combinaison des techniques traditionnelles de TLN avec les LLM pour créer des approches hybrides qui exploitent les forces des deux. En utilisant des méthodes traditionnelles de TLN pour des tâches comme le prétraitement de documents, le regroupement et l'extraction de sujets, vous pouvez efficacement organiser et structurer vos données textuelles. Ces informations structurées peuvent ensuite être transmises aux LLM pour des tâches plus avancées, comme la génération de résumés, la réponse aux questions ou la création de rapports complets.

Par exemple, considérons un cas d'utilisation où vous souhaitez générer un rapport de tendances pour un domaine spécifique basé sur un grand corpus de documents de tendances individuels. Au lieu de vous fier uniquement aux LLM, qui peuvent être coûteux en calcul et chronophages pour traiter de grands volumes de texte, vous pouvez employer une approche hybride :

1. Utiliser des techniques traditionnelles de TLN, comme la modélisation thématique (par exemple, l'Allocation de Dirichlet latente) ou les algorithmes de clustering (par exemple, K-moyennes), pour regrouper les documents de tendances similaires et identifier les thèmes et sujets clés au sein du corpus.

2. Alimenter les documents regroupés et les sujets identifiés dans un LLM, exploitant ses capacités supérieures de compréhension et de génération du langage pour créer des résumés cohérents et informatifs pour chaque groupe ou sujet.
3. Enfin, utiliser le LLM pour générer un rapport de tendances complet en combinant les résumés individuels, mettant en évidence les tendances les plus significatives et fournissant des insights et des recommandations basés sur les informations agrégées.

En combinant les techniques traditionnelles de TLN avec les LLM de cette manière, vous pouvez efficacement traiter de grandes quantités de données textuelles, extraire des insights significatifs et générer des rapports de haute qualité tout en optimisant les ressources computationnelles et les coûts.

Lorsque vous lancez dans vos projets de TALN, il est essentiel d'évaluer soigneusement les exigences et les contraintes spécifiques de chaque tâche et de réfléchir à la façon dont les méthodes traditionnelles de TALN et les LLMs peuvent être exploités conjointement pour obtenir les meilleurs résultats. En combinant l'efficacité et la précision des techniques traditionnelles avec la polyvalence et la puissance des LLMs, vous pouvez créer des solutions de TALN hautement efficaces et économiques qui apportent de la valeur à vos utilisateurs et parties prenantes.

Utilisation d'outils



Dans le domaine du développement d'applications basées sur l'IA, le concept d'"utilisation d'outils" ou d'"appel de fonction" s'est imposé comme une technique puissante permettant à votre LLM de se connecter à des outils externes, des APIs, des fonctions, des bases de données et d'autres ressources. Cette approche permet d'obtenir un ensemble de comportements plus riche que la simple sortie de texte, et des interactions plus dynamiques entre vos composants IA et le reste de l'écosystème de votre application. Comme nous l'examinerons dans ce chapitre, l'utilisation d'outils vous donne également la possibilité de faire générer des données par votre modèle d'IA de manière structurée.

Qu'est-ce que l'utilisation d'outils ?

L'utilisation d'outils, également connue sous le nom d'appel de fonction, est une technique qui permet aux développeurs de spécifier une liste de fonctions avec lesquelles un LLM peut interagir pendant le processus de génération. Ces outils peuvent aller de simples fonctions utilitaires à des APIs complexes ou des requêtes de base de données. En donnant au LLM accès à ces outils, les développeurs peuvent étendre les capacités du modèle et lui permettre d'effectuer des tâches nécessitant des connaissances ou des actions externes.

Figure 8. Exemple d'une définition de fonction pour un agent IA qui analyse des documents

```
1  FUNCTION = {  
2      name : "save_analysis",  
3      description : "Save analysis data for document",  
4      parameters : {  
5          type : "object",  
6          properties : {  
7              title : {  
8                  type : "string",  
9                  maxLength : 140  
10             },  
11             summary : {  
12                 type : "string",  
13                 description : "comprehensive multi-paragraph summary with  
14                     overview and list of sections (if applicable)"  
15             },  
16             tags : {  
17                 type : "array",  
18                 items : {  
19                     type : "string",  
20                     description : "lowercase tags representing main themes  
21                         of the document"  
22                 }  
23             }  
24         },  
25         "required" : %w[title summary tags]  
26     }
```

L'idée principale derrière l'utilisation d'outils est de donner au GML la capacité de sélectionner et d'exécuter dynamiquement les outils appropriés en fonction de l'entrée de l'utilisateur ou de la tâche à accomplir. Au lieu de s'appuyer uniquement sur les connaissances préentraînées du modèle, l'utilisation d'outils permet au GML d'exploiter des ressources externes pour générer des réponses plus précises, pertinentes et exploitables. L'utilisation d'outils rend des techniques comme la RAG (Génération augmentée par récupération) beaucoup plus faciles à mettre en œuvre qu'elles ne le seraient autrement.

Notez que sauf indication contraire, ce livre suppose que votre modèle d'IA n'a accès à aucun outil intégré côté serveur. Tous les outils que vous souhaitez mettre à la disposition de votre IA doivent être explicitement déclarés par vous dans chaque requête API, avec des dispositions pour gérer son exécution si et quand votre IA vous indique qu'elle souhaite utiliser cet outil dans sa réponse.

Le potentiel de l'utilisation d'outils

L'utilisation d'outils ouvre un large éventail de possibilités pour les applications basées sur l'IA. Voici quelques exemples de ce qui peut être réalisé avec l'utilisation d'outils :

- Agents conversationnels et assistants virtuels** : En connectant un GML à des outils externes, les agents conversationnels et les assistants virtuels peuvent effectuer des tâches plus complexes, comme récupérer des informations dans des bases de données, exécuter des appels API ou interagir avec d'autres systèmes. Par exemple, un agent conversationnel pourrait utiliser un outil CRM pour modifier le statut d'une affaire en fonction de la demande de l'utilisateur.

2. **Analyse de données et insights** : Les GML peuvent être connectés à des outils ou des bibliothèques d'analyse de données pour effectuer des tâches avancées de traitement de données. Cela permet aux applications de générer des insights, de mener des analyses comparatives ou de fournir des recommandations basées sur les données en réponse aux requêtes des utilisateurs.
3. **Recherche et récupération d'informations** : L'utilisation d'outils permet aux GML d'interagir avec des moteurs de recherche, des bases de données vectorielles ou d'autres systèmes de récupération d'informations. En transformant les requêtes utilisateur en requêtes de recherche, le GML peut récupérer des informations pertinentes à partir de multiples sources et fournir des réponses complètes aux questions des utilisateurs.
4. **Intégration avec des services externes** : L'utilisation d'outils permet une intégration transparente entre les applications basées sur l'IA et les services ou API externes. Par exemple, un GML pourrait interagir avec une API météo pour fournir des mises à jour météorologiques en temps réel ou une API de traduction pour générer des réponses multilingues.

Le flux de travail de l'utilisation d'outils

Le flux de travail de l'utilisation d'outils implique généralement quatre étapes clés :

1. Inclure les définitions de fonctions dans votre contexte de requête
2. Sélection dynamique (ou explicite) d'outils
3. Exécution de la ou des fonction(s)
4. Continuation optionnelle du prompt initial

Examinons chacune de ces étapes en détail.

Inclure les définitions de fonctions dans votre contexte de requête

L'IA sait quels outils sont à sa disposition car vous lui fournissez une liste dans le cadre de votre demande de complétion (généralement définie comme des fonctions utilisant une variante du schéma JSON).

La syntaxe précise de la définition des outils est spécifique au modèle.

Voici comment définir une fonction `get_weather` dans Claude 3 :

```
1  {
2      "name" : "get_weather",
3      "description" : "Get the current weather in a given location",
4      "input_schema" : {
5          "type" : "object",
6          "properties" : {
7              "location" : {
8                  "type" : "string",
9                  "description" : "The city and state, e.g. San Francisco, CA"
10             },
11             "unit" : {
12                 "type" : "string",
13                 "enum" : ["celsius", "fahrenheit"],
14                 "description" : "The unit of temperature"
15             }
16         },
17         "required" : ["location"]
18     }
19 }
```

Et voici comment vous définiriez la même fonction pour GPT-4, en la transmettant comme valeur du paramètre `tools` :

```
1  {
2      "name" : "get_current_weather",
3      "description" : "Get the current weather in a given location",
4      "parameters" : {
5          "type" : "object",
6          "properties" : {
7              "location" : {
8                  "type" : "string",
9                  "description" : "The city and state, e.g. San Francisco, CA",
10             },
11             "unit" : {
12                 "type" : "string",
13                 "enum" : ["celsius", "fahrenheit"],
14                 "description" : "The unit of temperature"
15             },
16         },
17         "required" : ["location"],
18     },
19 }
```

Presque pareil, mais différent sans raison apparente ! Comme c'est agaçant.

Les définitions de fonctions spécifient le nom, la description et les paramètres d'entrée. Les paramètres d'entrée peuvent être davantage définis en utilisant des attributs tels que les enums pour limiter les valeurs acceptables, et en spécifiant si un paramètre est requis ou non.

En plus des définitions de fonctions proprement dites, vous pouvez également inclure des instructions ou du contexte expliquant pourquoi et comment utiliser la fonction dans la directive système.

Par exemple, mon outil de recherche Web dans Olympia inclut cette directive système, qui rappelle à l'IA qu'elle dispose des outils mentionnés :

1 The `google_search` and `realtime_search` functions let you do research
2 on behalf of the user. In contrast to Google, realtime search is powered
3 by Perplexity and provides real-time information to curated current events
4 databases and news sources. Make sure to include URLs in your response so
5 user can do followup research.

Fournir des descriptions détaillées est considéré comme le facteur le plus important dans la performance des outils. Vos descriptions doivent expliquer chaque détail concernant l'outil, notamment :

- Ce que fait l'outil
- Quand il doit être utilisé (et quand il ne doit pas l'être)
- Ce que signifie chaque paramètre et comment il affecte le comportement de l'outil
- Toutes les mises en garde ou limitations importantes qui s'appliquent à l'implémentation de l'outil

Plus vous pourrez donner de contexte à l'IA concernant vos outils, mieux elle sera en mesure de décider quand et comment les utiliser. Par exemple, Anthropic recommande au moins 3 à 4 phrases par description d'outil pour sa série Claude 3, davantage si l'outil est complexe.

Ce n'est pas nécessairement intuitif, mais les descriptions sont également considérées comme plus importantes que les exemples. Bien que vous puissiez inclure des exemples d'utilisation d'un outil dans sa description ou dans le prompt associé, cela est moins important que d'avoir une explication claire et complète de l'objectif et des paramètres de l'outil. N'ajoutez des exemples qu'après avoir entièrement développé la description.

Voici un exemple de spécification de fonction API de type Stripe :

```
1  {
2      "name" : "createPayment",
3      "description" : "Create a new payment request",
4      "parameters" : {
5          "type" : "object",
6          "properties" : {
7              "transaction_amount" : {
8                  "type" : "number",
9                  "description" : "The amount to be paid"
10             },
11             "description" : {
12                 "type" : "string",
13                 "description" : "A brief description of the payment"
14             },
15             "payment_method_id" : {
16                 "type" : "string",
17                 "description" : "The payment method to be used"
18             },
19             "payer" : {
20                 "type" : "object",
21                 "description" : "Information about the payer, including their name,
22                               email, and identification number",
23                 "properties" : {
24                     "name" : {
25                         "type" : "string",
26                         "description" : "The payer's name"
27                     },
28                     "email" : {
29                         "type" : "string",
30                         "description" : "The payer's email address"
31                     },
32                     "identification" : {
33                         "type" : "object",
34                         "description" : "The payer's identification number",
35                         "properties" : {
36                             "type" : {
37                                 "type" : "string",
38                                 "description" : "Identification document (e.g. CPF, CNPJ)"
39                             },
40                             "number" : {
41                                 "type" : "string",
42                                 "description" : "The identification number"
43                         }
44                     }
45                 }
46             }
47         }
48     }
49 }
```

```
43         }
44     },
45     "required" : [ "type", "number" ]
46   }
47 },
48 "required" : [ "name", "email", "identification" ]
49 }
50 }
51 }
```



En pratique, certains modèles ont du mal à gérer les spécifications de fonctions imbriquées et à traiter des types de données complexes comme les tableaux, les dictionnaires, etc. Mais en théorie, vous devriez pouvoir fournir des spécifications JSON Schema de profondeur arbitraire !

Sélection Dynamique d'Outils

Lorsque vous exécutez une complétion de chat qui inclut des définitions d'outils, le LLM sélectionne dynamiquement le ou les outils les plus appropriés à utiliser et génère les paramètres d'entrée requis pour chaque outil.

En pratique, la capacité de l'IA à appeler *exactement* la bonne fonction et à suivre *exactement* votre spécification pour les entrées est aléatoire. Réduire le paramètre de température à 0.0 aide beaucoup, mais d'après mon expérience, vous aurez encore des erreurs occasionnelles. Ces échecs incluent des noms de fonctions hallucinés, des paramètres d'entrée mal nommés ou tout simplement manquants. Les paramètres sont transmis en JSON, ce qui signifie que vous verrez parfois des erreurs causées par du JSON tronqué, mal formaté ou autrement incorrect.



Les modèles d'[Auto-réparation des Données](#) peuvent aider à [réparer automatiquement](#) les appels de fonction qui échouent en raison d'erreurs de syntaxe.

Sélection Forcée (ou Explicite) d'Outils

Certains modèles vous donnent l'option de forcer l'appel d'une fonction particulière, en tant que paramètre dans la requête. Sinon, la décision d'appeler ou non la fonction est entièrement à la discrétion de l'IA.

La capacité de forcer un appel de fonction est cruciale dans certains scénarios où vous voulez vous assurer qu'un outil ou une fonction spécifique est exécuté, indépendamment du processus de sélection dynamique de l'IA. Il y a plusieurs raisons pour lesquelles cette capacité est importante :

1. **Contrôle Explicite** : Vous utilisez peut-être l'IA comme *Composant Discret* ou dans un flux de travail prédéfini qui nécessite l'exécution d'une fonction particulière à un moment précis. En forçant l'appel, vous pouvez garantir que la fonction souhaitée est invoquée au lieu de devoir demander gentiment à l'IA de le faire.
2. **Débogage et Tests** : Lors du développement et des tests d'applications basées sur l'IA, la capacité de forcer des appels de fonction est inestimable à des fins de débogage. En déclenchant explicitement des fonctions spécifiques, vous pouvez isoler et tester des composants individuels de votre application. Cela vous permet de vérifier l'exactitude des implémentations de fonctions, de valider les paramètres d'entrée et de vous assurer que les résultats attendus sont renvoyés.
3. **Gestion des Cas Limites** : Il peut y avoir des cas limites ou des scénarios exceptionnels où le processus de sélection dynamique de l'IA pourrait ne pas choisir d'exécuter une fonction qu'elle devrait, et vous le savez en fonction de processus externes. Dans de tels cas, avoir la capacité de forcer un appel de fonction vous permet de gérer ces situations explicitement. Définissez des règles ou des conditions dans la logique de votre application pour déterminer quand remplacer la discrétion de l'IA.
4. **Cohérence et Reproductibilité** : Si vous avez une séquence spécifique de fonctions qui doivent être exécutées dans un ordre particulier, forcer les appels

garantit que la même séquence est suivie à chaque fois. C'est particulièrement important dans les applications où la cohérence et le comportement prévisible sont critiques, comme dans les systèmes financiers ou les simulations scientifiques.

5. **Optimisation des Performances** : Dans certains cas, forcer un appel de fonction peut conduire à des optimisations de performances. Si vous savez qu'une fonction spécifique est requise pour une tâche particulière et que le processus de sélection dynamique de l'IA pourrait introduire une surcharge inutile, vous pouvez contourner le processus de sélection et invoquer directement la fonction requise. Cela peut aider à réduire la latence et améliorer l'efficacité globale de votre application.

En résumé, la capacité de forcer des appels de fonction dans les applications basées sur l'IA fournit un contrôle explicite, aide au débogage et aux tests, gère les cas limites, assure la cohérence et la reproductibilité. C'est un outil puissant dans votre arsenal, mais nous devons discuter d'un autre aspect de cette fonctionnalité importante.



Dans de nombreux cas d'utilisation décisionnels, nous voulons toujours que le modèle fasse un appel de fonction et ne voulons peut-être jamais que le modèle réponde uniquement avec ses connaissances internes. Par exemple, si vous routez entre plusieurs modèles spécialisés dans différentes tâches (entrée multilingue, mathématiques, etc.), vous pouvez utiliser le modèle d'appel de fonction pour déléguer les requêtes à l'un des modèles auxiliaires et ne jamais répondre indépendamment.

Paramètre de Choix d'Outil

GPT-4 et d'autres modèles de langage qui prennent en charge l'appel de fonction vous donnent un paramètre `tool_choice` pour contrôler si l'utilisation d'un outil est requise dans le cadre d'une complétion. Ce paramètre a trois valeurs possibles :

- `auto` donne à l'IA une totale discréption pour utiliser un outil ou simplement répondre
- `required` indique à l'IA qu'elle *doit* appeler un outil *au lieu* de répondre, mais laisse la sélection de l'outil à l'IA
- La troisième option consiste à définir le paramètre du `name_of_function` que vous souhaitez forcer. Plus de détails à ce sujet dans la section suivante.



Notez que si vous définissez `tool choice` comme `required`, le modèle sera forcé de choisir la fonction la plus pertinente parmi celles qui lui sont fournies, même si aucune ne correspond vraiment à la demande. Au moment de la publication, je ne connais aucun modèle qui renverrait une réponse `tool_calls` vide, ou qui utiliserait un autre moyen pour vous informer qu'il n'a pas trouvé de fonction appropriée à appeler.

Forcer un Appel de Fonction pour Obtenir une Sortie Structurée

La possibilité de forcer un appel de fonction vous offre un moyen d'obtenir des données structurées à partir d'une complétion de chat, au lieu d'avoir à les extraire vous-même de sa réponse en texte brut.

Pourquoi est-ce si important de forcer les fonctions pour obtenir des données structurées ? Tout simplement parce que l'extraction de données structurées à partir de la sortie d'un GML est un véritable casse-tête. Vous pouvez vous simplifier un peu la vie en demandant les données en XML, mais ensuite vous devez analyser le XML. Et que faites-vous quand ce XML est manquant parce que votre IA a répondu : "Je

suis désolé, mais je ne peux pas générer les données que vous avez demandées car bla, bla, bla..."

Lorsque vous utilisez les outils de cette manière :

- Vous devriez probablement définir un seul outil dans votre requête
- N'oubliez pas de forcer l'utilisation de sa fonction en utilisant le paramètre `tool_choice`
- Rappelez-vous que le modèle va transmettre l'entrée à l'outil, donc le nom de l'outil et sa description doivent être du point de vue du modèle, pas du vôtre

Ce dernier point mérite un exemple pour plus de clarté. Supposons que vous demandiez à l'IA de faire une analyse des sentiments sur un texte utilisateur. Le nom de la fonction ne serait pas `analyze_sentiment`, mais plutôt quelque chose comme `save_sentiment_analysis`. C'est l'IA qui fait l'analyse des sentiments, *pas l'outil*. Tout ce que fait l'outil (du point de vue de l'IA) est de sauvegarder les résultats de l'analyse.

Voici un exemple d'utilisation de Claude 3 pour enregistrer un résumé d'une image dans un JSON bien structuré, cette fois-ci depuis la ligne de commande en utilisant `curl` :

```
1 curl https://api.anthropic.com/v1/messages \
2   --header "content-type: application/json" \
3   --header "x-api-key: $ANTHROPIC_API_KEY" \
4   --header "anthropic-version: 2023-06-01" \
5   --header "anthropic-beta: tools-2024-04-04" \
6   --data \
7   '{
8     "model": "claude-3-sonnet-20240229",
9     "max_tokens": 1024,
10    "tools": [
11      {
12        "name": "record_summary",
13        "description": "Record summary of image into well-structured JSON.",
14        "input_schema": {
15          "type": "object",
16          "properties": {
17            "image": {
18              "type": "string",
19              "format": "uri"
20            }
21          }
22        }
23      }
24    ]
25  }'
```

```
15      "properties" : {
16          "key_colors" : {
17              "type" : "array",
18              "items" : {
19                  "type" : "object",
20                  "properties" : {
21                      "r" : {
22                          "type" : "number",
23                          "description" : "red value [0.0, 1.0]"
24                      },
25                      "g" : {
26                          "type" : "number",
27                          "description" : "green value [0.0, 1.0]"
28                      },
29                      "b" : {
30                          "type" : "number",
31                          "description" : "blue value [0.0, 1.0]"
32                      },
33                      "name" : {
34                          "type" : "string",
35                          "description" : "Human-readable color name
36                                         in snake_case, e.g.
37                                         \"olive_green\" or
38                                         \"turquoise\""
39                      }
40                  },
41                  "required" : [ "r", "g", "b", "name" ]
42              },
43              "description" : "Key colors in the image. Four or less."
44          },
45          "description" : {
46              "type" : "string",
47              "description" : "Image description. 1-2 sentences max."
48          },
49          "estimated_year" : {
50              "type" : "integer",
51              "description" : "Estimated year that the image was taken,
52                                         if is it a photo. Only set this if the
53                                         image appears to be non-fictional.
54                                         Rough estimates are okay !"
55          }
56      },
```

```
57         "required" : [ "key_colors", "description" ]
58     }
59 ],
60 "messages" : [
61     {
62         "role" : "user",
63         "content" : [
64             {
65                 "type" : "image",
66                 "source" : {
67                     "type" : "base64",
68                     "media_type" : "'$IMAGE_MEDIA_TYPE'",
69                     "data" : "'$IMAGE_BASE64'"
70                 }
71             },
72             {
73                 "type" : "text",
74                 "text" : "Use `record_summary` to describe this image."
75             }
76         ]
77     }
78 ]
79 }'
```

Dans l'exemple fourni, nous utilisons le modèle Claude 3 d'Anthropic pour générer un résumé JSON structuré d'une image. Voici comment cela fonctionne :

1. Nous définissons un seul outil nommé `record_summary` dans le tableau `tools` de la charge utile de la requête. Cet outil est chargé d'enregistrer un résumé de l'image dans un format JSON bien structuré.
2. L'outil `record_summary` possède un `input_schema` qui spécifie la structure attendue de la sortie JSON. Il définit trois propriétés :
 - `key_colors` : Un tableau d'objets représentant les couleurs principales de l'image. Chaque objet couleur possède des propriétés pour les valeurs rouge, vert et bleu (allant de 0.0 à 1.0) et un nom de couleur lisible par l'humain au format `snake_case`.

- **description** : Une propriété de type chaîne de caractères pour une brève description de l'image, limitée à 1-2 phrases.
 - **estimated_year** : Une propriété entière optionnelle pour l'année estimée de prise de l'image, si celle-ci semble être une photo non fictionnelle.
3. Dans le tableau `messages`, nous fournissons les données de l'image sous forme de chaîne encodée en base64 avec le type de média. Cela permet au modèle de traiter l'image dans le cadre de l'entrée.
 4. Nous demandons également à Claude d'utiliser l'outil `record_summary` pour décrire l'image.
 5. Lorsque la requête est envoyée au modèle Claude 3, celui-ci analyse l'image et génère un résumé JSON basé sur l'`input_schema` spécifié. Le modèle extrait les couleurs principales, fournit une brève description et estime l'année de prise de vue (le cas échéant).
 6. Le résumé JSON généré est transmis comme paramètres à l'outil `record_summary`, fournissant une représentation structurée des caractéristiques principales de l'image.

En utilisant l'outil `record_summary` avec un `input_schema` bien défini, nous pouvons obtenir un résumé JSON structuré d'une image sans dépendre de l'extraction de texte brut. Cette approche garantit que la sortie suit un format cohérent et peut être facilement analysée et traitée par les composants en aval de l'application.

La capacité à forcer un appel de fonction et à spécifier la structure de sortie attendue est une fonctionnalité puissante de l'utilisation d'outils dans les applications basées sur l'IA. Elle permet aux développeurs d'avoir plus de contrôle sur la sortie générée et simplifie l'intégration des données générées par l'IA dans le flux de travail de leur application.

Exécution de Fonction(s)

Vous avez défini des fonctions et interrogé votre IA, qui a décidé qu'elle devait appeler l'une de vos fonctions. Maintenant, c'est au tour de votre code d'application ou de votre

bibliothèque, si vous utilisez une gem Ruby comme `raix-rails`, de dispatcher l'appel de fonction et ses paramètres vers l'implémentation correspondante *dans votre code d'application*.

Votre code d'application décide quoi faire avec les résultats de l'exécution de la fonction. Peut-être que cela implique une seule ligne de code dans un lambda, ou peut-être que cela implique d'appeler une API externe. Peut-être que cela implique d'appeler un autre composant d'IA, ou peut-être que cela implique des centaines, voire des milliers de lignes de code dans le reste de votre système. C'est entièrement à vous de décider.

Parfois, l'appel de fonction est la fin de l'opération, mais si les résultats représentent des informations dans une chaîne de pensée qui doit être poursuivie par l'IA, alors votre code d'application doit insérer les résultats de l'exécution dans la transcription du chat et laisser l'IA continuer le traitement.

Par exemple, voici une déclaration de fonction `Raix` utilisée par l'AccountManager d'Olympia pour communiquer avec nos clients dans le cadre d'une Orchestration Intelligente des Flux de Travail pour le service client.

```
1 class AccountManager
2   include Raix ::ChatCompletion
3   include Raix ::FunctionDispatch
4
5   # lots of other functions...
6
7   function :notify_account_owner,
8     "Don't share UUID. Mention dollars if subscription changed",
9     message : { type : "string" } do |arguments|
10    account.owner.freeform_notify(
11      subject : "Account Change Notification",
12      message : arguments[ :message]
13    )
14    "Notified account owner"
15  end
```

Il n'est peut-être pas immédiatement évident de comprendre ce qui se passe ici, alors je vais le décomposer.

1. La classe `AccountManager` définit de nombreuses fonctions liées à la gestion des comptes. Elle peut modifier votre forfait, ajouter et supprimer des membres d'équipe, entre autres.
2. Ses instructions de haut niveau indiquent à `AccountManager` qu'il doit notifier le propriétaire du compte des résultats de la demande de modification du compte, en utilisant la fonction `notify_account_owner`.
3. La définition concise de la fonction comprend :
 - son nom
 - sa description
 - ses paramètres `message: { type: "string" }`
 - un bloc à exécuter lorsque la fonction est appelée

Après avoir mis à jour la transcription avec les résultats du bloc de fonction, la méthode `chat_completion` est appelée à nouveau. Cette méthode est chargée de renvoyer la transcription mise à jour de la conversation au modèle d'IA pour un traitement ultérieur. Nous appelons ce processus une *boucle de conversation*.

Lorsque le modèle d'IA reçoit une nouvelle demande de compléion de conversation avec une transcription mise à jour, il a accès aux résultats de la fonction précédemment exécutée. Il peut analyser ces résultats, les intégrer dans son processus de décision et générer la prochaine réponse ou action basée sur le contexte cumulatif de la conversation. Il peut choisir d'exécuter des fonctions supplémentaires basées sur le contexte mis à jour, ou il peut générer une réponse finale à la requête originale s'il détermine qu'aucun autre appel de fonction n'est nécessaire.

Poursuite optionnelle de la requête originale

Lorsque vous renvoyez les résultats de l'outil au GML et continuez le traitement de la requête originale, l'IA utilise ces résultats pour soit appeler des fonctions supplémentaires, soit générer une réponse finale en texte brut.



Certains modèles comme [Command-R](#) de Cohere peuvent citer les outils spécifiques qu'ils ont utilisés dans leurs réponses, offrant une transparence et une traçabilité supplémentaires.

Selon le modèle utilisé, les résultats de l'appel de fonction vivront dans des messages de transcription qui ont leur propre rôle spécial ou seront reflétés dans une autre syntaxe. Mais l'important est que ces données soient dans la transcription, afin qu'elles puissent être prises en compte par l'IA lorsqu'elle décide quoi faire ensuite.



Une erreur courante (et potentiellement coûteuse) est d'oublier d'ajouter les résultats de la fonction à la transcription avant de continuer la conversation. En conséquence, l'IA sera sollicitée essentiellement de la même manière qu'avant qu'elle n'appelle la fonction la première fois. En d'autres termes, du point de vue de l'IA, elle n'a pas encore appelé la fonction. Alors elle l'appelle à nouveau. Et encore. Et encore, indéfiniment jusqu'à ce que vous l'interrompiez. Espérons que votre contexte n'était pas trop grand et que votre modèle n'était pas trop coûteux !

Meilleures pratiques pour l'utilisation des outils

Pour tirer le meilleur parti de l'utilisation des outils, considérez les meilleures pratiques suivantes.

Définitions descriptives

Fournissez des noms et des descriptions clairs et descriptifs pour chaque outil et ses paramètres d'entrée. Cela aide le GML à mieux comprendre l'objectif et les capacités de chaque outil.

Je peux vous dire par expérience que la sagesse commune qui dit que “nommer est difficile” s’applique ici; j’ai vu des résultats radicalement différents des GMLs simplement en changeant les noms des fonctions ou la formulation des descriptions. Parfois, supprimer les descriptions *améliore* les performances.

Traitement des résultats d'outils

Lors de la transmission des résultats d'outils au GML, assurez-vous qu'ils sont bien structurés et complets. Utilisez des clés et des valeurs significatives pour représenter la sortie de chaque outil. Expérimitez avec différents formats et voyez lequel fonctionne le mieux, du JSON au texte brut.

Le [Result Interpreter](#) aborde ce défi en utilisant l'IA pour analyser les résultats et fournir des explications, des résumés ou des points clés compréhensibles par l'humain.

Gestion des erreurs

Mettez en place des mécanismes robustes de gestion des erreurs pour gérer les cas où le GML pourrait générer des paramètres d'entrée invalides ou non pris en charge pour les appels d'outils. Gérez et récupérez gracieusement toute erreur qui pourrait survenir pendant l'exécution de l'outil.

Une qualité particulièrement agréable de l'IA est qu'elle comprend les messages d'erreur! Ce qui signifie que si vous travaillez dans un état d'esprit rapide et approximatif, vous pouvez simplement capturer toutes les exceptions générées dans l'implémentation d'un outil et les renvoyer à l'IA pour qu'elle sache ce qui s'est passé!

Par exemple, voici une version simplifiée de l'implémentation de la recherche Google dans Olympia :

```
1  def google_search(conversation, params)
2    conversation.update_cstatus("Searching Google...")
3    query = params[ :query]
4    search = GoogleSearch.new(query).get_hash
5
6    conversation.update_cstatus("Summarizing results...")
7    SummarizeKnowledgeGraph.new.perform(conversation, search.to_json)
8  rescue StandardError => e
9    Honeybadger.notify(e)
10   { error : e.message }.inspect
11 end
```

Les recherches Google dans Olympia se font en deux étapes. D'abord vous effectuez la recherche, puis vous résumez les résultats. En cas d'échec, quelle qu'en soit la raison, le message d'erreur est encapsulé et renvoyé à l'IA. Cette technique est le fondement de pratiquement tous les modèles de *Gestion Intelligente des Erreurs*.

Par exemple, supposons que l'appel à l'API GoogleSearch échoue en raison d'une exception 503 Service Unavailable. Cette erreur remonte jusqu'au niveau supérieur du rescue, et la description de l'erreur est renvoyée à l'IA comme résultat de l'appel de fonction. Au lieu de simplement présenter à l'utilisateur un écran vide ou une erreur technique, l'IA dit quelque chose comme "Je suis désolé, mais je ne peux pas accéder à mes capacités de recherche Google pour le moment. Je peux réessayer plus tard, si vous le souhaitez."

Cela peut sembler n'être qu'une astuce intelligente, mais considérez un autre type d'erreur, où l'IA appelait une API externe et avait le contrôle direct des paramètres à transmettre à l'API. Peut-être a-t-elle fait une erreur dans la façon dont elle a générée ces paramètres ? À condition que le message d'erreur de l'API externe soit suffisamment détaillé, le fait de renvoyer le message d'erreur à l'IA appelante signifie qu'elle peut reconsidérer ces paramètres et réessayer. Automatiquement. Quelle que soit l'erreur.

Maintenant, pensez à ce qu'il faudrait pour reproduire ce type de gestion d'erreurs robuste dans du code *normal*. C'est pratiquement impossible.

Raffinement Itératif

Si le LLM ne recommande pas les outils appropriés ou génère des réponses sous-optimales, itérez sur les définitions d'outils, les descriptions et les paramètres d'entrée. Affinez et améliorez continuellement la configuration des outils en fonction du comportement observé et des résultats souhaités.

1. Commencez par des définitions d'outils simples : Commencez par définir des outils avec des noms, des descriptions et des paramètres d'entrée clairs et concis. Évitez de trop compliquer la configuration des outils au début et concentrez-vous sur les fonctionnalités essentielles. Par exemple, si vous voulez sauvegarder les résultats d'une analyse de sentiment, commencez par une définition basique comme :

```
1  {
2      "name" : "save_sentiment_score",
3      "description" : "Analyze user-provided text and generate sentiment score",
4      "parameters" : {
5          "type" : "object",
6          "properties" : {
7              "score" : {
8                  "type" : "float",
9                  "description" : "sentiment score from -1 (negative) to 1 (positive)"
10             }
11         },
12         "required" : ["score"]
13     }
14 }
```

2. Tester et observer : Une fois que vous avez mis en place les définitions d'outils initiales, testez-les avec différentes invites et observez comment le GML interagit avec l'outil. Portez attention à la qualité et à la pertinence des réponses générées. Si le GML génère des réponses sous-optimales, il est temps d'affiner les définitions d'outils.

3. Affiner les descriptions : Si le GML ne comprend pas correctement l'objectif d'un outil, essayez d'affiner la description de l'outil. Fournissez plus de contexte, d'exemples ou de clarifications pour guider le GML dans l'utilisation efficace de l'outil. Par exemple, vous pouvez mettre à jour la description de l'outil d'analyse des sentiments pour aborder plus spécifiquement le *ton émotionnel* du texte analysé :

```
1  {
2    "name" : "save_sentiment_score",
3    "description" : "Determine the overall emotional tone of a piece of text,
4    such as customer reviews, social media posts, or feedback comments.",
5    ...
6 }
```

4. Ajustez les paramètres d'entrée : Si le LLM génère des paramètres d'entrée non valides ou non pertinents pour un outil, envisagez d'ajuster les définitions des paramètres. Ajoutez des contraintes plus spécifiques, des règles de validation ou des exemples pour clarifier le format d'entrée attendu.
5. Itérez selon les retours : Surveillez en permanence la performance de vos outils et recueillez les retours des utilisateurs ou des parties prenantes. Utilisez ces retours pour identifier les domaines à améliorer et apportez des ajustements itératifs aux définitions des outils. Par exemple, si les utilisateurs signalent que l'analyse ne gère pas bien le sarcasme, vous pouvez ajouter une note dans la description :

```
1  {
2      "name" : "save_sentiment_score",
3      "description" : "Analyze the sentiment of a given text and return a sentiment
4          score between -1 (negative) and 1 (positive). Note : Sarcasm should be
5          considered negative.",
6      ...
7  }
```

En affinant itérativement vos définitions d'outils basées sur le comportement observé et les retours d'expérience, vous pouvez progressivement améliorer la performance et l'efficacité de votre application basée sur l'IA. N'oubliez pas de maintenir les définitions d'outils claires, concises et centrées sur la tâche spécifique à accomplir. Testez et validez régulièrement les interactions des outils pour vous assurer qu'elles correspondent à vos résultats souhaités.

Composition et Chaînage d'Outils

L'un des aspects les plus puissants de l'utilisation d'outils, qui n'a été qu'évoqué jusqu'à présent, est la capacité de composer et de chaîner plusieurs outils ensemble pour accomplir des tâches complexes. En concevant soigneusement vos définitions d'outils et leurs formats d'entrée/sortie, vous pouvez créer des blocs de construction réutilisables qui peuvent être combinés de différentes manières.

Prenons un exemple où vous construisez un pipeline d'analyse de données pour votre application basée sur l'IA. Vous pourriez avoir les outils suivants :

1. **DataRetrieval** : Un outil qui récupère des données depuis une base de données ou une API selon des critères spécifiés.
2. **DataProcessing** : Un outil qui effectue des calculs, des transformations ou des agrégations sur les données récupérées.
3. **DataVisualization** : Un outil qui présente les données traitées dans un format convivial, comme des graphiques ou des diagrammes.

En chaînant ces outils ensemble, vous pouvez créer un flux de travail puissant qui récupère les données pertinentes, les traite et présente les résultats de manière significative. Voici à quoi pourrait ressembler le flux de travail d'utilisation des outils :

1. Le GML reçoit une requête utilisateur demandant des informations sur les données de vente pour une catégorie de produits spécifique.
2. Le GML sélectionne l'outil `DataRetrieval` et génère les paramètres d'entrée appropriés pour récupérer les données de vente pertinentes de la base de données.
3. Les données récupérées sont “transmises” à l'outil `DataProcessing`, qui calcule des métriques telles que le revenu total, le prix de vente moyen et le taux de croissance.
4. Les données traitées sont ensuite digérées par l'outil `DataVisualization`, qui crée un graphique ou un diagramme visuellement attrayant pour représenter les informations, renvoyant l'URL du graphique au GML.
5. Enfin, le GML génère une réponse formatée à la requête de l'utilisateur en utilisant `markdown`, incorporant les données visualisées et fournissant un résumé des principales conclusions.

En composant ces outils ensemble, vous pouvez créer un flux de travail d'analyse de données fluide qui peut être facilement intégré dans votre application. La beauté de cette approche est que chaque outil peut être développé et testé indépendamment, puis combiné de différentes manières pour résoudre divers problèmes.

Pour permettre une composition et un chaînage fluides des outils, il est important de définir des formats d'entrée et de sortie clairs pour chaque outil.

Par exemple, l'outil `DataRetrieval` pourrait accepter des paramètres tels que les détails de connexion à la base de données, le nom de la table et les conditions de requête, et renvoyer l'ensemble de résultats sous forme d'objet JSON structuré. L'outil `DataProcessing` peut alors s'attendre à recevoir cet objet JSON en entrée et produire un objet JSON transformé en sortie. En standardisant le flux de données entre les outils, vous pouvez assurer la compatibilité et la réutilisabilité.

Lors de la conception de votre écosystème d'outils, réfléchissez à la façon dont différents outils peuvent être combinés pour répondre aux cas d'utilisation courants de votre application. Envisagez de créer des outils de haut niveau qui encapsulent les flux de travail courants ou la logique métier, facilitant ainsi la sélection et l'utilisation efficace par le GML.

N'oubliez pas que la puissance de l'utilisation d'outils réside dans la flexibilité et la modularité qu'elle offre. En décomposant des tâches complexes en outils plus petits et réutilisables, vous pouvez créer une application basée sur l'IA robuste et adaptable capable de relever un large éventail de défis.

Orientations Futures

À mesure que le domaine du développement d'applications basées sur l'IA évolue, nous pouvons nous attendre à de nouvelles avancées dans les capacités d'utilisation d'outils. Voici quelques orientations futures potentielles :

1. **Utilisation d'outils à plusieurs étapes** : Les GML pourront peut-être décider combien de fois ils doivent utiliser les outils pour générer une réponse satisfaisante. Cela pourrait impliquer plusieurs cycles de sélection et d'exécution d'outils basés sur des résultats intermédiaires.
2. **Outils prédefinis** : Les plateformes d'IA pourront fournir un ensemble d'outils prédefinis que les développeurs pourront utiliser directement, comme des interpréteurs Python, des outils de recherche web ou des fonctions utilitaires courantes.
3. **Intégration transparente** : À mesure que l'utilisation d'outils se généralise, nous pouvons nous attendre à une meilleure intégration entre les plateformes d'IA et les cadres de développement populaires, facilitant l'incorporation de l'utilisation d'outils dans leurs applications.

L'utilisation d'outils est une technique puissante qui permet aux développeurs d'exploiter tout le potentiel des GML dans les applications basées sur l'IA. En connectant les GML à des outils et des ressources externes, vous pouvez créer des systèmes plus dynamiques, intelligents et conscients du contexte qui peuvent s'adapter aux besoins des utilisateurs et fournir des informations et des actions précieuses.

Bien que l'utilisation d'outils offre d'immenses possibilités, il est important d'être conscient des défis et des considérations potentiels. Un aspect clé est la gestion de la complexité des interactions entre outils et la garantie de la stabilité et de la fiabilité du système global. Vous devez gérer les scénarios où les appels d'outils peuvent échouer, renvoyer des résultats inattendus ou avoir des implications sur les performances. De plus, vous devez envisager des mesures de sécurité et de contrôle d'accès pour empêcher l'utilisation non autorisée ou malveillante des outils. Des mécanismes appropriés de gestion des erreurs, de journalisation et de surveillance sont cruciaux pour maintenir l'intégrité et les performances de votre application basée sur l'IA.

En explorant les possibilités d'utilisation d'outils dans vos propres projets, n'oubliez pas de commencer par des objectifs clairs, de concevoir des définitions d'outils bien structurées, et d'itérer en fonction des retours d'expérience et des résultats. Avec l'approche et l'état d'esprit appropriés, l'utilisation d'outils peut débloquer de nouveaux niveaux d'innovation et de valeur dans vos applications basées sur l'IA.

Traitements de flux



La diffusion de données via HTTP, également connue sous le nom d'événements envoyés par le serveur (SSE), est un mécanisme où le serveur envoie continuellement des données au client au fur et à mesure qu'elles deviennent disponibles, sans que le client n'ait à les demander explicitement. Comme la réponse de l'IA est générée de manière incrémentielle, il est logique d'offrir une expérience utilisateur réactive en affichant la sortie de l'IA au fur et à mesure de sa génération. Et en fait, toutes les API des fournisseurs d'IA que je connais proposent les réponses en streaming comme option dans leurs points de terminaison de completion.

La raison pour laquelle ce chapitre apparaît ici dans le livre, juste après [Utilisation des outils](#), est liée à la puissance de la combinaison de l'utilisation d'outils avec les réponses en direct de l'IA aux utilisateurs. Cela permet des expériences dynamiques et interactives où l'IA peut traiter les entrées des utilisateurs, utiliser divers outils et fonctions à sa discrédition, puis fournir des réponses en temps réel.

Pour obtenir cette interaction fluide, vous devez écrire des gestionnaires de flux qui peuvent distribuer les appels de fonctions d'outils invoqués par l'IA ainsi que la sortie en texte brut à l'utilisateur final. La nécessité de boucler après le traitement d'une fonction d'outil ajoute un défi intéressant au travail.

Implémentation d'un ReplyStream

Pour démontrer comment le traitement de flux peut être implémenté, ce chapitre va examiner en profondeur une version simplifiée de la classe `ReplyStream` qui est utilisée dans Olympia. Les instances de cette classe peuvent être passées comme paramètre `stream` dans les bibliothèques clientes d'IA telles que `ruby-openai` et `openrouter`.

Voici comment j'utilise `ReplyStream` dans le `PromptSubscriber` d'Olympia, qui écoute via Wisper la création de nouveaux messages utilisateur.

```
1  class PromptSubscriber
2    include Raix ::ChatCompletion
3    include Raix ::PromptDeclarations
4
5    # many other declarations omitted...
6
7    prompt text : -> { user_message.content },
8      stream : -> { ReplyStream.new(self) },
9      until : -> { bot_message.complete? }
10
11  def message_created(message) # invoked by Wisper
12    return unless message.role.user? && message.content?
13
14  # rest of the implementation omitted...
```

En plus d'une référence context vers l'abonné de prompt qui l'a instanciée, la classe `ReplyStream` possède également des variables d'instance pour stocker un tampon de données reçues, ainsi que des tableaux pour suivre les noms de fonctions et les arguments invoqués pendant le traitement du flux.

```
1  class ReplyStream
2    attr_accessor :buffer, :f_name, :f_arguments, :context
3
4    delegate :bot_message, :dispatch, to: :context
5
6    def initialize(context)
7      self.context = context
8      self.buffer = []
9      self.f_name = []
10     self.f_arguments = []
11   end
12
13  def call(chunk, bytesize = nil)
14    # ...
15  end
16
17  # ...
18 end
```

La méthode `initialize` configure l'état initial de l'instance `ReplyStream`, en initialisant le tampon, le contexte et d'autres variables.

La méthode `call` est le point d'entrée principal pour le traitement des données en flux continu. Elle prend un fragment de données `chunk` (représenté sous forme de table de hachage) et un paramètre optionnel `bytesize`, qui dans notre exemple n'est pas utilisé. À l'intérieur de cette méthode, la classe utilise la correspondance de motifs pour gérer différents scénarios selon la structure du fragment reçu.



L'appel de `deep_symbolize_keys` sur le fragment permet de rendre la correspondance de motifs plus élégante, en nous permettant de travailler avec des symboles plutôt que des chaînes de caractères.

```
1  def call(chunk, _bytesize)
2      case chunk.deep_symbolize_keys
3
4      in { # match function name
5          choices : [
6              {
7                  delta : {
8                      tool_calls : [
9                          { index : index, function : {name : name} }
10                     ]
11                 }
12             }
13         ]
14
15         f_name[index] = name
```

Le premier motif que nous cherchons à identifier est un appel d'outil ainsi que le nom de fonction qui lui est associé. Si nous en détectons un, nous le plaçons dans le tableau `f_name`. Nous stockons les noms de fonction dans un tableau indexé, car le modèle est capable d'effectuer des appels de fonction en parallèle, envoyant plusieurs fonctions à exécuter simultanément.

L'appel de fonctions en parallèle est la capacité d'un modèle d'IA à effectuer plusieurs appels de fonction ensemble, permettant aux effets et aux résultats de ces appels de fonction d'être résolus en parallèle. Cela est particulièrement utile lorsque les fonctions prennent beaucoup de temps, et réduit les allers-retours avec l'API, ce qui peut en retour permettre d'économiser une quantité significative de jetons.

Ensuite, nous devons identifier les arguments correspondant aux appels de fonction.

```

1  in { # match arguments
2    choices : [
3      {
4        delta : {
5          tool_calls : [
6            {
7              index : index, function : {arguments : argument }
8            }
9          ]
10        }
11      }
12    ]}
13
14  f_arguments[index] ||= "" # initialize if not already
15  f_arguments[index] << argument

```

De la même manière que nous avons géré les noms de fonction, nous rangeons les arguments dans un tableau indexé.

Ensuite, nous recherchons les messages destinés à l'utilisateur, qui arriveront du serveur un jeton à la fois et seront assignés à la variable `new_content`. Nous devons également surveiller `finish_reason`. Il restera `nil` jusqu'au dernier fragment de la séquence de sortie.

```

1  in {
2    choices : [
3      { delta : {content : new_content}, finish_reason : finish_reason }
4    ]}
5
6    # you could transmit every chunk to the user here...
7    buffer << new_content.to_s
8
9    if finish_reason.present?
10      finalize
11    elsif new_content.to_s.match?(/\n\n/)
12      send_to_client # ...or buffer and transmit once per paragraph
13    end

```

De manière importante, nous ajoutons une expression de correspondance de motif pour gérer les messages d'erreur envoyés par le fournisseur du modèle d'IA. Dans

les environnements de développement locaux, nous levons une exception, mais en production, nous journalisons l'erreur et finalisons.

```
1  in { error : { message : } }
2    if Rails.env.local ?
3      raise message
4    else
5      Honeybadger.notify("AI Error : #{message}")
6      finalize
7  end
```

La dernière clause else de l'instruction case s'exécute si aucun des motifs précédents n'a correspondu. C'est simplement une mesure de sécurité pour que nous puissions découvrir si le modèle d'IA commence à nous envoyer des fragments non reconnus.

```
1  else
2    Honeybadger.notify("Unrecognized Chunk : #{chunk}")
3  end
4 end
```

La méthode `send_to_client` est responsable de l'envoi du contenu mis en tampon vers le client. Elle vérifie que le tampon n'est pas vide, met à jour le contenu du message du bot, effectue le rendu du message du bot et sauvegarde le contenu dans la base de données pour assurer la persistance des données.

```

1 def send_to_client
2   # no need to process pure whitespace
3   return if buffer.join.squish.blank?
4
5   # set the buffer content on the bot message
6   content = buffer.join
7   bot_message.content = content
8
9   # save to database so that we never lose data
10  # even if the stream doesn't terminate correctly
11  bot_message.update_column( :content, content)
12
13  # update content via websocket
14  ConversationRenderer.update(bot_message)
15 end

```

La méthode `finalize` est appelée lorsque le traitement du flux est terminé. Elle distribue les appels de fonction si certains ont été reçus pendant le flux, met à jour le message du bot avec le contenu final et d'autres informations pertinentes, et réinitialise l'historique des appels de fonction

```

1 def finalize
2   if f_name.any?
3     f_name.each_with_index do |name, index|
4       # takes care of calling the function wherever it's implemented
5       dispatch(name :, arguments : JSON.parse(f_arguments[index]))
6     end
7
8     # reset the function call history
9     f_name.clear
10    f_arguments.clear
11  else
12    content = buffer.join.presence
13    bot_message.update !(content :, complete : true)
14    ConversationRenderer.update(bot_message)
15  end
16 end

```

Si le modèle décide d'appeler une fonction, vous devez “distribuer” cet appel de fonction

(nom et arguments) de manière à ce qu'il soit exécuté et que les messages `function_call` et `function_result` soient ajoutés à la transcription de la conversation

D'après mon expérience, il est préférable de gérer la création des messages de fonction à un seul endroit dans votre base de code, plutôt que de s'appuyer sur les implémentations des outils. C'est non seulement plus propre, mais il y a aussi une raison pratique très importante : si le modèle d'IA appelle une fonction et ne voit pas les messages d'appel et de résultat correspondants dans la transcription lors de la boucle, *il appellera la même fonction à nouveau*. Potentiellement indéfiniment. Rappelez-vous que l'IA est complètement sans état, donc à moins que vous ne lui renvoyiez ces appels de fonction, ils n'ont jamais eu lieu.

```
1 # PromptSubscriber#dispatch
2
3 def dispatch(name :, arguments :)
4   # adds a function_call message to the conversation transcript
5   # plus dispatches to tool and returns result
6   conversation.function_call !(name, arguments).then do |result|
7     # add function result message to the transcript
8     conversation.function_result !(name, result)
9   end
10 end
```



Effacer l'historique des appels de fonction après leur exécution est tout aussi important que de s'assurer que l'appel et les résultats se retrouvent dans votre transcription, afin d'éviter de continuer à appeler les mêmes fonctions encore et encore à chaque boucle.

La « Boucle de Conversation »

Je mentionne souvent le bouclage, mais si vous débutez avec l'appel de fonctions, il n'est peut-être pas évident de comprendre *pourquoi* nous avons besoin d'une boucle. La raison

est que lorsque l'IA « demande » d'exécuter des fonctions d'outil en son nom, elle cesse de répondre. C'est à vous d'exécuter ces fonctions, de rassembler les résultats, de les ajouter à la transcription, puis de soumettre à nouveau l'invite originale afin d'obtenir un nouveau lot d'appels de fonction ou des résultats destinés à l'utilisateur.

Dans la classe `PromptSubscriber`, nous utilisons la méthode `prompt` du module `PromptDeclarations` pour définir le comportement de la boucle de conversation. Le paramètre `until` est défini sur `-> { bot_message.complete? }`, ce qui signifie que la boucle continuera jusqu'à ce que le `bot_message` soit marqué comme complet.

```
1  prompt text : -> { user_message.content },
2      stream : -> { ReplyStream.new(self) },
3      until : -> { bot_message.complete? }
```



Mais quand est-ce que `bot_message` est marqué comme complet? Si vous avez oublié, reportez-vous à la ligne 13 de la méthode `finalize`.

Passons en revue toute la logique de traitement de flux.

1. Le `PromptSubscriber` reçoit un nouveau message utilisateur via la méthode `message_created`, qui est invoquée par le système pub/sub Wisper chaque fois que l'utilisateur final crée une nouvelle invite.
2. La méthode de classe `prompt` définit de manière déclarative le comportement de la logique de complétion du chat pour le `PromptSubscriber`. Le modèle d'IA exécutera une complétion de chat avec le contenu du message de l'utilisateur, une nouvelle instance de `ReplyStream` comme paramètre de flux, et la condition de boucle spécifiée.
3. Le modèle d'IA traite l'invite et commence à générer une réponse. Au fur et à mesure que la réponse est diffusée en flux, la méthode `call` de l'instance `ReplyStream` est invoquée pour chaque fragment de données.

4. Si le modèle d'IA décide d'appeler une fonction d'outil, le nom de la fonction et les arguments sont extraits du fragment et stockés respectivement dans les tableaux `f_name` et `f_arguments`.
5. Si le modèle d'IA génère du contenu destiné à l'utilisateur, il est mis en mémoire tampon et envoyé au client via la méthode `send_to_client`.
6. Une fois le traitement du flux terminé, la méthode `finalize` est appelée. Si des fonctions d'outil ont été invoquées pendant le flux, elles sont expédiées en utilisant la méthode `dispatch` du `PromptSubscriber`.
7. La méthode `dispatch` ajoute un message `function_call` à la transcription de la conversation, exécute la fonction d'outil correspondante, et ajoute un message `function_result` à la transcription avec le résultat de l'appel de fonction.
8. Après l'expédition des fonctions d'outil, l'historique des appels de fonction est effacé pour éviter les appels de fonction en double dans les boucles suivantes.
9. Si aucune fonction d'outil n'a été invoquée, la méthode `finalize` met à jour le `bot_message` avec le contenu final, le marque comme complet, et envoie le message mis à jour au client.
10. La condition de boucle `-> { bot_message.complete? }` est évaluée. Si le `bot_message` n'est pas marqué comme complet, la boucle continue, et l'invite originale est soumise à nouveau avec la transcription de conversation mise à jour.
11. Les étapes 3 à 10 sont répétées jusqu'à ce que le `bot_message` soit marqué comme complet, indiquant que le modèle d'IA a terminé de générer sa réponse et qu'aucune autre fonction d'outil ne doit être exécutée.

En implémentant cette boucle de conversation, vous permettez au modèle d'IA d'engager une interaction bidirectionnelle avec l'application, exécutant des fonctions d'outil selon les besoins et générant des réponses destinées à l'utilisateur jusqu'à ce que la conversation atteigne une conclusion naturelle.

La combinaison du traitement de flux et de la boucle de conversation permet des expériences dynamiques et interactives basées sur l'IA, où le modèle d'IA peut traiter

les entrées utilisateur, utiliser divers outils et fonctions, et fournir des réponses en temps réel basées sur le contexte évolutif de la conversation.

Auto-continuation

Il est important d'être conscient des limitations des sorties d'IA. La plupart des modèles ont un nombre maximum de jetons qu'ils peuvent générer dans une seule réponse, déterminé par le paramètre `max_tokens`. Si le modèle d'IA atteint cette limite lors de la génération d'une réponse, il s'arrêtera brusquement et indiquera que la sortie a été tronquée.

Dans la réponse en flux de l'API de la plateforme d'IA, vous pouvez détecter cette situation en examinant la variable `finish_reason` dans le fragment. Si la `finish_reason` est définie sur "length" (ou une autre valeur clé spécifique au modèle), cela signifie que le modèle a atteint sa limite maximale de jetons pendant la génération et que la sortie a été interrompue.

Une façon de gérer ce scénario avec élégance et de fournir une expérience utilisateur fluide est d'implémenter un mécanisme d'auto-continuation dans votre logique de traitement de flux. En ajoutant une correspondance de motif pour les raisons de fin liées à la longueur, vous pouvez choisir de boucler et de continuer automatiquement la sortie là où elle s'est arrêtée.

Voici un exemple volontairement simplifié de la façon dont vous pouvez modifier la méthode `call` dans la classe `ReplyStream` pour prendre en charge l'auto-continuation :

```
1 LENGTH_STOPS = %w[length MAX_TOKENS]
2
3 def call(chunk, _bytesize)
4   case chunk.deep_symbolize_keys
5     # ...
6
7   in {
8     choices : [
9       { delta : {content : new_content},
10         finish_reason : finish_reason } ] }
11
12   buffer << new_content.to_s
13
14   if finish_reason.blank?
15     send_to_client if new_content.to_s.match?(/\n\n/)
16   elsif LENGTH_STOPS.include?(finish_reason)
17     continue_cutoff
18   else
19     finalize
20   end
21
22   # ...
23 end
24 end
25
26 private
27
28 def continue_cutoff
29   conversation.bot_message!(buffer.join, visible : false)
30   conversation.user_message!("please continue", visible : false)
31   bot_message.update_column( :created_at, Time.current)
32 end
```

Dans cette version modifiée, lorsque la `finish_reason` indique une sortie tronquée, au lieu de finaliser le flux, nous ajoutons une paire de messages à la transcription sans la finaliser, déplaçons le message de réponse original visible par l'utilisateur vers le "bas" de la transcription en mettant à jour son attribut `created_at`, puis laissons la boucle se poursuivre, permettant ainsi à l'IA de continuer là où elle s'était arrêtée.

Rappelez-vous que le point de terminaison de complétion de l'IA est sans état. Il ne

“connaît” que ce que vous lui communiquez via la transcription. Dans ce cas, la façon dont nous communiquons à l’IA qu’elle a été interrompue est d’ajouter des messages “invisibles” (pour l’utilisateur final) à la transcription. Souvenez-vous cependant qu’il s’agit d’un exemple volontairement simplifié. Une implémentation réelle nécessiterait une gestion plus approfondie de la transcription pour s’assurer que nous ne gaspillons pas de jetons et/ou ne confondons pas l’IA avec des messages d’assistant dupliqués dans la transcription.

Une véritable implémentation de l’auto-continuation devrait également inclure ce qu’on appelle une “logique de disjoncteur” pour empêcher les boucles incontrôlées. La raison en est que, compte tenu de certains types de prompts utilisateur et de paramètres `max_tokens` bas, l’IA pourrait continuer à produire des sorties visibles par l’utilisateur en boucle sans fin.

Gardez à l’esprit que chaque boucle nécessite une requête séparée, et que chaque requête consomme à nouveau l’intégralité de votre transcription. Vous devriez certainement prendre en compte les compromis entre l’expérience utilisateur et l’utilisation de l’API lorsque vous décidez d’implémenter ou non l’auto-continuation dans votre application. L’auto-continuation peut être particulièrement dangereusement coûteuse, surtout lors de l’utilisation de modèles commerciaux premium.

Conclusion

Le traitement de flux est un aspect crucial de la construction d’applications basées sur l’IA qui combinent l’utilisation d’outils avec des réponses d’IA en direct. En gérant efficacement les données en streaming des API de plateformes d’IA, vous pouvez fournir une expérience utilisateur fluide et interactive, gérer des réponses volumineuses,

optimiser l'utilisation des ressources et gérer les erreurs avec élégance.

La classe `Conversation::ReplyStream` fournie démontre comment le traitement de flux peut être implémenté dans une application Ruby en utilisant la correspondance de motifs et l'architecture événementielle. En comprenant et en tirant parti des techniques de traitement de flux, vous pouvez libérer tout le potentiel de l'intégration de l'IA dans vos applications et offrir des expériences utilisateur puissantes et engageantes.

Données auto-réparatrices



Les données auto-réparatrices constituent une approche puissante pour garantir l'intégrité, la cohérence et la qualité des données dans les applications en exploitant les capacités des grands modèles de langage (LLM). Cette catégorie de modèles se concentre sur l'idée d'utiliser l'IA pour détecter, diagnostiquer et corriger automatiquement les anomalies, les incohérences ou les erreurs dans les données, réduisant ainsi la charge de travail des développeurs tout en maintenant un niveau élevé de fiabilité des données.

À la base, les modèles de données auto-réparatrices reconnaissent que les données sont l'élément vital de toute application, et garantir leur exactitude et leur intégrité est crucial pour le bon fonctionnement et l'expérience utilisateur de l'application. Cependant, la gestion et le maintien de la qualité des données peuvent être une tâche complexe et chronophage, en particulier lorsque les applications grandissent en taille et en complexité. C'est là que la puissance de l'IA entre en jeu.

Dans les modèles de données auto-réparatrices, des agents IA sont employés pour surveiller et analyser en permanence les données de votre application. Ces modèles ont la capacité de comprendre et d'interpréter les modèles, les relations et les anomalies au sein des données. En exploitant leurs capacités de traitement et de compréhension du langage naturel, ils peuvent identifier les problèmes potentiels ou les incohérences dans les données et prendre les mesures appropriées pour les corriger.

Le processus de données auto-réparatrices implique généralement plusieurs étapes clés :

- 1. Surveillance des données** : Les agents IA surveillent constamment les flux de données, les bases de données ou les systèmes de stockage de l'application, à la recherche de signes d'anomalies, d'incohérences ou d'erreurs. Alternativement, vous pouvez activer un composant IA en réaction à une exception.
- 2. Détection d'anomalies** : Lorsqu'un problème est détecté, l'agent IA analyse les données en détail pour identifier la nature et la portée spécifiques du problème. Cela peut impliquer la détection de valeurs manquantes, de formats incohérents ou de données qui violent des règles ou des contraintes prédefinies.
- 3. Diagnostic et correction** : Une fois le problème identifié, l'agent IA utilise ses connaissances et sa compréhension du domaine des données pour déterminer la ligne de conduite appropriée. Cela peut impliquer la correction automatique des données, le remplissage des valeurs manquantes ou le signalement du problème pour une intervention humaine si nécessaire.
- 4. Apprentissage continu (optionnel, selon le cas d'utilisation)** : À mesure que votre agent IA rencontre et résout divers problèmes de données, il peut produire des descriptions de ce qui s'est passé et de la façon dont il a réagi. Ces métadonnées peuvent être intégrées dans des processus d'apprentissage qui vous permettent (et peut-être au modèle sous-jacent, via l'ajustement fin) de devenir plus efficace au fil du temps dans l'identification et la résolution des anomalies de données.

En détectant et en corrigent automatiquement les problèmes de données, vous pouvez garantir que votre application fonctionne avec des données fiables et de haute qualité.

Cela réduit le risque d'erreurs, d'incohérences ou de bogues liés aux données affectant la fonctionnalité ou l'expérience utilisateur de l'application.

Une fois que vous avez des agents IA gérant la tâche de surveillance et de correction des données, vous pouvez concentrer vos efforts sur d'autres aspects critiques de l'application. Cela permet d'économiser du temps et des ressources qui auraient autrement été consacrés au nettoyage et à la maintenance manuelle des données. En fait, à mesure que vos applications grandissent en taille et en complexité, la gestion manuelle de la qualité des données devient de plus en plus difficile. Les modèles de "Données auto-réparatrices" s'adaptent efficacement en exploitant la puissance de l'IA pour gérer de grands volumes de données et détecter les problèmes en temps réel.



De par leur nature, les modèles d'IA peuvent s'adapter aux changements de modèles de données, de schémas ou d'exigences au fil du temps avec peu ou pas de supervision. Tant que leurs directives fournissent des orientations adéquates, en particulier concernant les résultats attendus, votre application peut être capable d'évoluer et de gérer de nouveaux scénarios de données sans nécessiter d'intervention manuelle extensive ou de modifications de code.

Les modèles de données auto-réparatrices s'alignent bien avec les autres catégories de modèles que nous avons discutées, comme la "Multitude d'agents". La capacité de données auto-réparatrices peut être considérée comme un type spécialisé d'agent qui se concentre spécifiquement sur la garantie de la qualité et de l'intégrité des données. Ce type d'agent fonctionne aux côtés d'autres agents IA, chacun contribuant à différents aspects de la fonctionnalité de l'application.

La mise en œuvre des modèles de données auto-réparatrices dans la pratique nécessite une conception et une intégration soignées des modèles d'IA dans l'architecture de l'application. En raison des risques de perte et de corruption des données, vous devez définir des directives claires sur la façon dont vous utiliserez cette technique. Vous devez

également prendre en compte des facteurs tels que la performance, l'évolutivité et la sécurité des données.

Étude de cas pratique : Réparation du JSON défectueux

L'une des façons les plus pratiques et pratiques d'exploiter les données auto-réparatrices est également très simple à expliquer : la réparation du JSON défectueux.

Cette technique peut être appliquée au défi courant de traiter des données imparfaites ou incohérentes générées par les LLM, comme du JSON défectueux, et fournit une approche pour détecter et corriger automatiquement ces problèmes.

Chez Olympia, je rencontre régulièrement des scénarios où les LLM génèrent des données JSON qui ne sont pas parfaitement valides. Cela peut se produire pour diverses raisons, comme lorsque le LLM ajoute des commentaires avant ou après le code JSON, ou introduit des erreurs de syntaxe telles que des virgules manquantes ou des guillemets doubles non échappés. Ces problèmes peuvent entraîner des erreurs d'analyse syntaxique et provoquer des perturbations dans le fonctionnement de l'application.

Pour résoudre ce problème, j'ai mis en œuvre une solution pratique sous la forme d'une classe `JsonFixer`. Cette classe incarne le modèle de "Données auto-réparables" en prenant le JSON défectueux comme entrée et en utilisant un LLM pour le corriger tout en préservant autant que possible les informations et l'intention d'origine.

```
1  class JsonFixer
2    include Raix ::ChatCompletion
3
4    def call(bad_json, error_message)
5      raise "No data provided" if bad_json.blank? || error_message.blank?
6
7      transcript << {
8        system : "Consider user-provided JSON that generated a parse
9                  exception. Do your best to fix it while preserving the
10                 original content and intent as much as possible."
11 transcript << { user : bad_json }
12 transcript << { assistant : "What is the error message ?" }
13 transcript << { user : error_message }
14 transcript << { assistant : "Here is the corrected JSON\n```\njson\n```\n" }
15
16      self.stop = ["```"]
17
18      chat_completion(json : true)
19    end
20
21    def model
22      "mistralai/mixtral-8x7b-instruct :nitro"
23    end
24  end
```



Notez comment `JsonFixer` utilise [Ventriloquist](#) pour guider les réponses de l'IA.

Le processus d'auto-réparation des données JSON fonctionne comme suit :

1. **Génération JSON** : Un LLM est utilisé pour générer des données JSON basées sur certains prompts ou exigences. Cependant, en raison de la nature des LLM, le JSON généré n'est pas toujours parfaitement valide. L'analyseur JSON lèvera bien sûr une erreur `ParserError` si vous lui fournissez du JSON invalide.

```
1 begin
2   JSON.parse(llm_generated_json)
3 rescue JSON::ParserError => e
4   JSONFixer.new.call(llm_generated_json, e.message)
5 end
```

Notez que le message d'erreur est également transmis à l'appel `JSONFixer` afin qu'il n'ait pas à faire des suppositions complètes sur ce qui ne va pas avec les données, d'autant plus que l'analyseur syntaxique vous dira souvent exactement ce qui ne va pas.

2. **Correction basée sur les LLM** : La classe `JSONFixer` renvoie le JSON défectueux à un LLM, accompagné d'une instruction ou d'une invite spécifique pour corriger le JSON tout en préservant autant que possible l'information et l'intention originales. Le LLM, entraîné sur de vastes quantités de données et comprenant la syntaxe JSON, tente de corriger les erreurs et de générer une chaîne JSON valide. La [Délimitation des réponses](#) est utilisée pour contraindre la sortie du LLM, et nous choisissons Mixtral 8x7B comme modèle d'IA, car il est particulièrement adapté à ce type de tâche.
3. **Validation et intégration** : La chaîne JSON corrigée renvoyée par le LLM est analysée par la classe `JSONFixer` elle-même, car elle a appelé `chat_completion(json: true)`. Si le JSON corrigé passe la validation, il est réintégré dans le flux de travail de l'application, permettant à celle-ci de continuer à traiter les données de manière transparente. Le mauvais JSON a été “guéri”.

Bien que j'aie écrit et réécrit ma propre implémentation de `JSONFixer` à plusieurs reprises, je doute que le temps total investi dans toutes ces versions dépasse une heure ou deux.

Notez que la préservation de l'intention est un élément clé de tout modèle de données auto-réparantes. Le processus de correction basé sur les LLM vise à préserver autant que possible l'information et l'intention originales du JSON généré. Cela garantit que le

JSON corrigé conserve sa signification sémantique et peut être utilisé efficacement dans le contexte de l'application.

Cette mise en œuvre pratique de l'approche “Données auto-réparantes” dans Olympia démontre clairement comment l'IA, en particulier les LLM, peut être exploitée pour résoudre des défis réels liés aux données. Elle illustre la puissance de la combinaison des techniques de programmation traditionnelles avec les capacités de l'IA pour construire des applications robustes et efficaces.

La Loi de Postel et le Modèle des “Données Auto-réparantes”

Les “Données auto-réparantes”, comme illustrées par la classe JSONFixer, s'alignent bien avec le principe connu sous le nom de Loi de Postel, également appelé Principe de robustesse. La Loi de Postel stipule :

“Soyez conservateur dans ce que vous faites, soyez libéral dans ce que vous acceptez des autres.”

Ce principe, initialement articulé par Jon Postel, un pionnier des débuts d'Internet, souligne l'importance de construire des systèmes qui sont tolérants aux entrées diverses ou même légèrement incorrectes tout en maintenant une stricte adhésion aux protocoles spécifiés lors de l'envoi des sorties.

Dans le contexte des “Données auto-réparantes”, la classe JSONFixer incarne la Loi de Postel en étant libérale dans l'acceptation des données JSON défectueuses ou imparfaites générées par les LLM. Elle ne rejette pas immédiatement et n'échoue pas lorsqu'elle rencontre du JSON qui n'adhère pas strictement au format attendu. Au lieu de cela, elle adopte une approche tolérante et tente de corriger le JSON en utilisant la puissance des LLM.

En étant libérale dans l'acceptation de JSON imparfait, la classe JSONFixer démontre

sa robustesse et sa flexibilité. Elle reconnaît que les données dans le monde réel se présentent souvent sous diverses formes et peuvent ne pas toujours être conformes aux spécifications strictes. En gérant et en corrigeant gracieusement ces écarts, la classe garantit que l'application peut continuer à fonctionner sans heurts, même en présence de données imparfaites.

D'autre part, la classe JSONFixer adhère également à l'aspect conservateur de la Loi de Postel en ce qui concerne la sortie. Après avoir corrigé le JSON à l'aide des LLM, la classe valide le JSON corrigé pour s'assurer qu'il est strictement conforme au format attendu. Elle maintient l'intégrité et l'exactitude des données avant de les transmettre à d'autres parties de l'application. Cette approche conservatrice garantit que la sortie de la classe JSONFixer est fiable et cohérente, favorisant l'interopérabilité et empêchant la propagation d'erreurs.

Anecdotes intéressantes sur Jon Postel :

- Jon Postel (1943-1998) était un informaticien américain qui a joué un rôle crucial dans le développement d'Internet. Il était connu comme le "Dieu d'Internet" pour ses contributions significatives aux protocoles et standards sous-jacents.
- Postel était l'éditeur de la série de documents Request for Comments (RFC), une série de notes techniques et organisationnelles sur Internet. Il a rédigé ou co-rédigé plus de 200 RFC, y compris les protocoles fondamentaux tels que TCP, IP et SMTP.
- En plus de ses contributions techniques, Postel était connu pour son approche humble et collaborative. Il croyait en l'importance d'atteindre un consensus et de travailler ensemble pour construire un réseau robuste et interopérable.
- Postel a occupé le poste de Directeur de la Division des réseaux informatiques à l'Institut des Sciences de l'Information (ISI) de l'Université de Californie du Sud (USC) de 1977 jusqu'à son décès prématuré en 1998.
- En reconnaissance de ses immenses contributions, Postel a reçu à titre posthume le prestigieux prix Turing en 1998, souvent considéré comme le

“Prix Nobel de l’informatique.”

La classe `JSONFixer` favorise la robustesse, la flexibilité et l’interopérabilité, qui étaient des valeurs fondamentales que Postel a défendues tout au long de sa carrière. En construisant des systèmes qui tolèrent les imperfections tout en maintenant une stricte adhérence aux protocoles, nous pouvons créer des applications plus résilientes et adaptables face aux défis du monde réel.

Considérations et Contre-indications

L’applicabilité des approches de données auto-réparantes dépend entièrement du type de données que votre application traite. Il y a une raison pour laquelle vous pourriez ne pas vouloir simplement modifier dynamiquement `JSON.parse` pour corriger automatiquement *toutes les erreurs d’analyse JSON* dans votre application : toutes les erreurs ne peuvent pas ou ne devraient pas être corrigées automatiquement.

L’auto-réparation est particulièrement délicate lorsqu’elle est associée à des exigences réglementaires ou de conformité liées au traitement et à la gestion des données. Certains secteurs, comme la santé et la finance, ont des réglementations si strictes concernant l’intégrité des données et la traçabilité que toute correction de données en “boîte noire” sans supervision ou journalisation appropriée peut violer ces réglementations. Il est crucial de s’assurer que toutes les techniques de données auto-réparantes que vous développez sont conformes aux cadres légaux et réglementaires applicables.

L’application des techniques de données auto-réparantes, en particulier celles impliquant des modèles d’IA, peut également avoir un impact important sur les performances de l’application et l’utilisation des ressources. Le traitement de grands volumes de données par des modèles d’IA pour la détection et la correction d’erreurs peut être intensif en calcul. Il est important d’évaluer les compromis entre les avantages

des données auto-réparantes et les coûts associés en termes de performance et de ressources.

Cela dit, plongeons dans les facteurs impliqués dans la décision de quand et où appliquer cette approche puissante.

Criticité des Données

Lors de l'examen de l'application des techniques de données auto-réparantes, il est crucial d'évaluer la criticité des données traitées. Le niveau de criticité fait référence à l'importance et à la sensibilité des données dans le contexte de votre application et de son domaine d'activité.

Dans certains cas, la correction automatique des erreurs de données peut ne pas être appropriée, en particulier si les données sont très sensibles ou ont des implications juridiques. Par exemple, considérez les scénarios suivants :

1. **Transactions Financières** : Dans les applications financières, comme les systèmes bancaires ou les plateformes de trading, la précision des données est de la plus haute importance. Même des erreurs mineures dans les données financières peuvent avoir des conséquences importantes, comme des soldes de compte incorrects, des fonds mal acheminés ou des décisions de trading erronées. Dans ces cas, les corrections automatisées sans vérification et audit approfondis peuvent introduire des risques inacceptables.
2. **Dossiers Médicaux** : Les applications de santé traitent des données de patients hautement sensibles et confidentielles. Les inexactitudes dans les dossiers médicaux peuvent avoir de graves implications pour la sécurité des patients et les décisions de traitement. La modification automatique des données médicales sans supervision et validation appropriées par des professionnels de santé qualifiés peut violer les exigences réglementaires et mettre en danger le bien-être des patients.

3. Documents Juridiques : Les applications traitant des documents juridiques, tels que les contrats, les accords ou les dépôts judiciaires, nécessitent une précision et une intégrité strictes. Même des erreurs mineures dans les données juridiques peuvent avoir des conséquences juridiques importantes. Les corrections automatisées dans ce domaine peuvent ne pas être appropriées, car les données nécessitent souvent un examen manuel et une vérification par des experts juridiques pour garantir leur validité et leur force exécutoire.

Dans ces scénarios de données critiques, les risques associés aux corrections automatisées l'emportent souvent sur les avantages potentiels. Les conséquences de l'introduction d'erreurs ou de la modification incorrecte des données peuvent être graves, entraînant des pertes financières, des responsabilités juridiques, ou même des préjudices pour les individus.

Lorsqu'on traite des données hautement critiques, il est essentiel de privilégier les processus de vérification et de validation manuels. La supervision et l'expertise humaines sont cruciales pour garantir l'exactitude et l'intégrité des données. Les techniques d'auto-réparation automatisées peuvent toujours être utilisées pour signaler des erreurs ou des incohérences potentielles, mais la décision finale sur les corrections doit impliquer le jugement et l'approbation humains.

Cependant, il est important de noter que toutes les données d'une application n'ont pas nécessairement le même niveau de criticité. Au sein d'une même application, il peut y avoir des sous-ensembles de données moins sensibles ou ayant un impact moindre en cas d'erreurs. Dans ces cas, les techniques de données auto-réparantes peuvent être appliquées sélectivement à ces sous-ensembles de données spécifiques, tandis que les données critiques restent soumises à une vérification manuelle.

L'essentiel est d'évaluer soigneusement la criticité de chaque catégorie de données dans votre application et de définir des directives et des processus clairs pour gérer les corrections en fonction des risques et des implications associés. En différenciant les données critiques (c'est-à-dire les grands livres, les dossiers médicaux) des données

non critiques (c'est-à-dire les adresses postales, les avertissements de ressources), vous pouvez trouver un équilibre entre l'exploitation des avantages des techniques de données auto-réparantes lorsque c'est approprié et le maintien d'un contrôle et d'une supervision stricts lorsque c'est nécessaire.

En fin de compte, la décision d'appliquer des techniques de données auto-réparantes aux données critiques doit être prise en consultation avec des experts du domaine, des conseillers juridiques et d'autres parties prenantes pertinentes. Il est essentiel de prendre en compte les exigences spécifiques, les réglementations et les risques associés aux données de votre application et d'aligner les stratégies de correction des données en conséquence.

Gravité des Erreurs

Lors de l'application des techniques de données auto-réparantes, il est important d'évaluer la gravité et l'impact des erreurs de données. Toutes les erreurs ne sont pas égales, et la ligne de conduite appropriée peut varier en fonction de la gravité du problème.

Les incohérences mineures ou les problèmes de formatage peuvent convenir à une correction automatique. Par exemple, un système de données auto-réparantes chargé de corriger du JSON défectueux peut gérer les virgules manquantes ou les guillemets doubles non échappés sans altérer significativement le sens ou la structure des données. Ces types d'erreurs sont souvent simples à corriger et ont un impact minimal sur l'intégrité globale des données.

Cependant, les erreurs plus graves qui modifient fondamentalement le sens ou l'intégrité des données peuvent nécessiter une approche différente. Dans de tels cas, les corrections automatisées peuvent s'avérer insuffisantes, et une intervention humaine peut être nécessaire pour garantir l'exactitude et la validité des données.

C'est là qu'intervient le concept d'utilisation de l'IA elle-même pour aider à déterminer la gravité des erreurs. En tirant parti des capacités des modèles d'IA, nous pouvons

concevoir des agents de données auto-réparateurs qui non seulement corrigent les erreurs, mais évaluent également leur gravité et prennent des décisions éclairées sur la façon de les traiter.

Par exemple, considérons un agent de données auto-réparateur chargé de corriger les incohérences dans les données qui alimentent une base de données clients. L'agent peut être conçu pour analyser les données et identifier les erreurs potentielles, telles que des informations manquantes ou contradictoires. Cependant, au lieu de corriger automatiquement toutes les erreurs, l'agent peut être équipé d'appels d'outils supplémentaires lui permettant de signaler les erreurs graves pour un examen humain.

Voici un exemple de la façon dont cela peut être mis en œuvre :

```
1  class CustomerDataReviewer
2    include Raix ::ChatCompletion
3    include Raix ::FunctionDeclarations
4
5    attr_accessor :customer
6
7    function :flag_for_review, reason : { type : "string" } do |params|
8      AdminNotifier.review_request(customer, params[ :reason])
9    end
10
11   def initialize(customer)
12     self.customer = customer
13   end
14
15   def call(customer_data)
16     transcript << {
17       system : "You are a customer data reviewer. Your task is to identify
18       and correct inconsistencies in customer data.
19
20       < additional instructions here... >
21
22       If you encounter severe errors that require human review, use the
23       `flag_for_review` tool to flag the data for manual intervention."
24     }
25     transcript << { user : customer.to_json }
26     transcript << { assistant : "Reviewed/corrected data :\n```\njson\n" }
```

```
27
28     self.stop = ["~`~`"]
29
30     chat_completion(json : true).then do |result|
31         return if result.blank?
32
33         customer.update(result)
34     end
35 end
36 end
```

Dans cet exemple, l'agent `CustomerDataHealer` est conçu pour identifier et corriger les incohérences dans les données clients. Une fois de plus, nous utilisons `Response Fencing` et `Ventiloquist` pour obtenir une sortie structurée. Il est important de noter que la directive système de l'agent inclut des instructions pour utiliser la fonction `flag_for_review` si des erreurs graves sont rencontrées.

Lorsque l'agent traite les données clients, il analyse les données et tente de corriger toute incohérence. Si l'agent détermine que les erreurs sont graves et nécessitent une intervention humaine, il peut utiliser l'outil `flag_for_review` pour signaler les données et fournir une raison pour ce signalement.

La méthode `chat_completion` est appelée avec `json: true` pour analyser les données clients corrigées au format JSON. Il n'y a pas de disposition pour une boucle après un appel de fonction, donc le résultat sera vide si `flag_for_review` a été invoqué. Sinon, les données du client sont mises à jour avec les données examinées et potentiellement corrigées.

En intégrant l'évaluation de la gravité des erreurs et l'option de signaler les données pour un examen humain, l'agent de données auto-réparantes devient plus intelligent et adaptable. Il peut gérer les erreurs mineures automatiquement tout en escaladant les erreurs graves vers des experts humains pour une intervention manuelle.

Les critères spécifiques pour déterminer la gravité des erreurs peuvent être définis dans la directive de l'agent en fonction des connaissances du domaine et des exigences

commerciales. Des facteurs tels que l'impact sur l'intégrité des données, le risque de perte ou de corruption des données, et les conséquences de données incorrectes peuvent être pris en compte lors de l'évaluation de la gravité.

En utilisant l'IA pour évaluer la gravité des erreurs et en fournissant des options pour l'intervention humaine, les techniques de données auto-réparantes peuvent trouver un équilibre entre l'automatisation et le maintien de l'exactitude des données. Cette approche garantit que les erreurs mineures sont corrigées efficacement tandis que les erreurs graves reçoivent l'attention et l'expertise nécessaires des examinateurs humains.

Complexité du domaine

Lors de l'examen de l'application des techniques de données auto-réparantes, il est important d'évaluer la complexité du domaine de données et les règles régissant sa structure et ses relations. La complexité du domaine peut avoir un impact significatif sur l'efficacité et la faisabilité des approches de correction automatique des données.

Les techniques de données auto-réparantes fonctionnent bien lorsque les données suivent des modèles et des contraintes bien définis. Dans les domaines où la structure des données est relativement simple et les relations entre les éléments de données sont directes, les corrections automatisées peuvent être appliquées avec un haut degré de confiance. Par exemple, la correction des problèmes de formatage ou l'application de contraintes de type de données de base peuvent souvent être gérées efficacement par des agents de données auto-réparantes.

Cependant, à mesure que la complexité du domaine de données augmente, les défis associés à la correction automatique des données augmentent également. Dans les domaines avec une logique métier complexe, des relations complexes entre les entités de données, ou des règles et exceptions spécifiques au domaine, les techniques de données auto-réparantes peuvent ne pas toujours capturer les nuances et peuvent introduire des conséquences imprévues.

Prenez l'exemple d'un domaine complexe : un système de trading financier. Dans ce domaine, les données impliquent divers instruments financiers, données de marché, règles de trading et exigences réglementaires. Les relations entre les différents éléments de données peuvent être complexes, et les règles régissant la validité et la cohérence des données peuvent être hautement spécifiques au domaine.

Dans un domaine aussi complexe, un agent de données auto-réparantes chargé de corriger les incohérences dans les données de trading devrait avoir une compréhension approfondie des règles et contraintes spécifiques au domaine. Il devrait prendre en compte des facteurs tels que les réglementations du marché, les limites de trading, les calculs de risque et les procédures de règlement. Les corrections automatisées dans ce contexte peuvent ne pas toujours capturer toute la complexité du domaine et peuvent involontairement introduire des erreurs ou violer des règles spécifiques au domaine.

Pour relever les défis de la complexité du domaine, les techniques de données auto-réparantes peuvent être améliorées en incorporant des connaissances et des règles spécifiques au domaine dans les modèles d'IA et les agents. Cela peut être réalisé grâce à des techniques telles que :

- 1. Formation spécifique au domaine :** Les modèles d'IA utilisés pour les données auto-réparantes peuvent être dirigés ou même affinés sur des ensembles de données spécifiques au domaine qui capturent les subtilités et les règles du domaine particulier. En exposant les modèles à des données et scénarios représentatifs, ils peuvent apprendre les modèles, contraintes et exceptions spécifiques au domaine.
- 2. Contraintes basées sur des règles :** Les agents de données auto-réparantes peuvent être augmentés avec des contraintes explicites basées sur des règles qui encodent les connaissances spécifiques au domaine. Ces règles peuvent être définies par des experts du domaine et intégrées dans le processus de correction des données. Les modèles d'IA peuvent alors utiliser ces règles pour guider leurs décisions et assurer la conformité aux exigences spécifiques du domaine.

3. **Collaboration avec les experts du domaine** : Dans les domaines complexes, il est crucial d'impliquer les experts du domaine dans la conception et le développement des techniques de données auto-réparantes. Les experts du domaine peuvent fournir des informations précieuses sur les subtilités des données, les règles métier et les cas limites potentiels. Leurs connaissances peuvent être incorporées dans les modèles d'IA et les agents pour améliorer la précision et la fiabilité des corrections automatiques des données en utilisant les modèles [Human In The Loop](#).
4. **Approche incrémentale et itérative** : Lorsqu'on traite des domaines complexes, il est souvent bénéfique d'adopter une approche incrémentale et itérative pour les données auto-réparantes. Au lieu d'essayer d'automatiser les corrections pour l'ensemble du domaine d'un seul coup, concentrez-vous sur des sous-domaines ou des catégories de données spécifiques où les règles et contraintes sont bien comprises. Élargissez progressivement la portée des techniques auto-réparantes à mesure que la compréhension du domaine s'accroît et que les techniques prouvent leur efficacité.

En tenant compte de la complexité du domaine des données et en intégrant les connaissances spécifiques au domaine dans les techniques de données auto-correctrices, vous pouvez trouver un équilibre entre l'automatisation et la précision. Il est important de reconnaître que les données auto-correctrices ne constituent pas une solution universelle et que l'approche doit être adaptée aux exigences et aux défis spécifiques de chaque domaine.

Dans les domaines complexes, une approche hybride combinant les techniques de données auto-correctrices avec l'expertise humaine et la supervision peut s'avérer plus efficace. Les corrections automatisées peuvent gérer les cas courants et bien définis, tandis que les scénarios complexes ou les exceptions peuvent être signalés pour examen et intervention humaine. Cette approche collaborative garantit que les avantages de l'automatisation sont réalisés tout en maintenant le contrôle et la précision nécessaires dans les domaines de données complexes.

Explicabilité et Transparence

L'explicabilité fait référence à la capacité de comprendre et d'interpréter le raisonnement derrière les décisions prises par les modèles d'IA, tandis que la transparence implique d'offrir une visibilité claire sur le processus de correction des données.

Dans de nombreux contextes, les modifications de données doivent être auditables et justifiables. Les parties prenantes, y compris les utilisateurs métier, les auditeurs et les organismes de réglementation, peuvent exiger des explications sur les raisons pour lesquelles certaines corrections de données ont été effectuées et comment les modèles d'IA sont parvenus à ces décisions. Cela est particulièrement crucial dans les domaines où l'exactitude et l'intégrité des données ont des implications importantes, comme la finance, la santé et les questions juridiques.

Pour répondre au besoin d'explicabilité et de transparence, les techniques de données auto-correctrices doivent intégrer des mécanismes qui fournissent des aperçus du processus décisionnel des modèles d'IA. Cela peut être réalisé par diverses approches :

- 1. Chaîne de pensée** : Demander au modèle d'expliquer son raisonnement “à voix haute” avant d'appliquer des modifications aux données peut permettre une meilleure compréhension du processus décisionnel et peut générer des explications lisibles par l'humain pour les corrections effectuées. Le compromis est une légère complexité supplémentaire dans la séparation de l'explication de la sortie de données structurées, qui peut être résolue par...
- 2. Génération d'explications** : Les agents de données auto-correctrices peuvent être équipés de la capacité de générer des explications lisibles par l'humain pour les corrections qu'ils effectuent. Cela peut être réalisé en demandant au modèle de produire son processus décisionnel sous forme d'explications facilement compréhensibles *intégrées dans les données elles-mêmes*. Par exemple, un agent de données auto-correctrices pourrait générer un rapport qui met en évidence les incohérences spécifiques des données qu'il a identifiées, les corrections qu'il a

appliquées et la logique derrière ces corrections.

3. **Importance des caractéristiques** : Les modèles d'IA peuvent être instruits avec des informations sur l'importance des différentes caractéristiques ou attributs dans le processus de correction des données dans le cadre de leurs directives. Ces directives, à leur tour, peuvent être exposées aux parties prenantes humaines. En identifiant les facteurs clés qui influencent les décisions du modèle, les parties prenantes peuvent comprendre le raisonnement derrière les corrections et évaluer leur validité.
4. **Journalisation et audit** : La mise en œuvre de mécanismes complets de journalisation et d'audit est cruciale pour maintenir la transparence dans le processus de données auto-correctrices. Chaque correction de données effectuée par les modèles d'IA doit être enregistrée, y compris les données originales, les données corrigées et les actions spécifiques entreprises. Cette piste d'audit permet une analyse rétrospective et fournit un enregistrement clair des modifications apportées aux données.
5. **Approche avec intervention humaine** : L'incorporation d'une approche avec intervention humaine peut améliorer l'explicabilité et la transparence des techniques de données auto-correctrices. En impliquant des experts humains dans l'examen et la validation des corrections générées par l'IA, les organisations peuvent s'assurer que les corrections sont alignées avec les connaissances du domaine et les exigences métier. La supervision humaine ajoute une couche supplémentaire de responsabilité et permet d'identifier tout biais ou erreur potentiel dans les modèles d'IA.
6. **Surveillance et évaluation continues** : La surveillance et l'évaluation régulières des performances des techniques de données auto-correctrices sont essentielles pour maintenir la transparence et la confiance. En évaluant la précision et l'efficacité des modèles d'IA au fil du temps, les organisations peuvent identifier les écarts ou les anomalies et prendre des mesures correctives. La surveillance continue aide à garantir que le processus de données auto-correctrices reste fiable

et aligné avec les résultats souhaités.

L'explicabilité et la transparence sont des considérations cruciales lors de la mise en œuvre des techniques de données auto-correctrices. En fournissant des explications claires pour les corrections de données, en maintenant des pistes d'audit complètes et en impliquant la supervision humaine, les organisations peuvent établir la confiance dans le processus de données auto-correctrices et s'assurer que les modifications apportées aux données sont justifiables et alignées avec les objectifs commerciaux.

Il est important de trouver un équilibre entre les avantages de l'automatisation et le besoin de transparence. Bien que les techniques de données auto-correctrices puissent améliorer significativement la qualité et l'efficacité des données, elles ne doivent pas se faire au détriment de la visibilité et du contrôle sur le processus de correction des données. En concevant des agents de données auto-correctrices avec l'explicabilité et la transparence à l'esprit, les organisations peuvent exploiter la puissance de l'IA tout en maintenant le niveau nécessaire de responsabilité et de confiance dans les données.

Conséquences imprévues

Bien que les techniques de données auto-correctrices visent à améliorer la qualité et la cohérence des données, il est crucial d'être conscient des conséquences imprévues potentielles. Les corrections automatisées, si elles ne sont pas soigneusement conçues et surveillées, peuvent modifier par inadvertance le sens ou le contexte des données, entraînant des problèmes en aval.

L'un des principaux risques des données auto-correctrices est l'introduction de biais ou d'erreurs dans le processus de correction des données. Les modèles d'IA, comme tout autre système logiciel, peuvent être soumis à des biais présents dans les données d'entraînement ou introduits par la conception des algorithmes. Si ces biais ne sont pas identifiés et atténués, ils peuvent se propager à travers le processus de données auto-correctrices et entraîner des modifications de données biaisées ou incorrectes.

Par exemple, considérons un travailleur de données auto-réparateur chargé de corriger les incohérences dans les données démographiques des clients. Si le modèle d'IA a appris des biais à partir de données historiques, comme l'association de certaines professions ou niveaux de revenus à des genres ou ethnies spécifiques, il pourrait faire des suppositions incorrectes et modifier les données d'une manière qui renforce ces biais. Cela peut conduire à des profils clients inexacts, des décisions commerciales mal orientées et des résultats potentiellement discriminatoires.

Une autre conséquence imprévue potentielle est la perte d'informations précieuses ou de contexte pendant le processus de correction des données. Les techniques d'auto-réparation des données se concentrent souvent sur la standardisation et la normalisation des données pour assurer la cohérence. Cependant, dans certains cas, les données originales peuvent contenir des nuances, des exceptions ou des informations contextuelles importantes pour comprendre l'ensemble de la situation. Les corrections automatisées qui appliquent aveuglément la standardisation peuvent involontairement supprimer ou masquer ces informations précieuses.

Par exemple, imaginons un travailleur de données auto-réparateur responsable de la correction d'incohérences dans les dossiers médicaux. Si le travailleur rencontre l'historique médical d'un patient présentant une condition rare ou un plan de traitement inhabituel, il pourrait tenter de normaliser les données pour les faire correspondre à un modèle plus courant. Cependant, ce faisant, il pourrait perdre les détails spécifiques et le contexte qui sont cruciaux pour représenter précisément la situation unique du patient. Cette perte d'information peut avoir de graves implications pour les soins aux patients et la prise de décision médicale.

Pour atténuer les risques de conséquences imprévues, il est essentiel d'adopter une approche proactive lors de la conception et de la mise en œuvre des techniques d'auto-réparation des données :

1. **Tests et validation approfondis** : Avant de déployer des travailleurs de données auto-réparateurs en production, il est crucial de tester et de valider

minutieusement leur comportement dans diverses situations. Cela inclut des tests avec des jeux de données représentatifs couvrant divers cas limites, exceptions et biais potentiels. Des tests rigoureux aident à identifier et à traiter toute conséquence imprévue avant qu'elle n'affecte les données réelles.

2. **Surveillance et évaluation continues** : La mise en place de mécanismes de surveillance et d'évaluation continues est essentielle pour détecter et atténuer les conséquences imprévues au fil du temps. L'examen régulier des résultats des processus d'auto-réparation des données, l'analyse de l'impact sur les systèmes en aval et la prise de décision, ainsi que la collecte des retours des parties prenantes peuvent aider à identifier tout effet négatif et à déclencher des actions correctives en temps opportun. Si votre organisation dispose de tableaux de bord opérationnels, l'ajout de métriques clairement visibles liées aux modifications automatisées des données est probablement une bonne idée. Ajouter des alarmes connectées aux grandes déviations par rapport à l'activité normale de modification des données est probablement une encore meilleure idée !
3. **Supervision et intervention humaines** : Maintenir une supervision humaine et la capacité d'intervenir dans le processus d'auto-réparation des données est crucial. Bien que l'automatisation puisse grandement améliorer l'efficacité, il est important que des experts humains examinent et valident les corrections effectuées par les modèles d'IA, en particulier dans les domaines critiques ou sensibles. Le jugement humain et l'expertise du domaine peuvent aider à identifier et à traiter toute conséquence imprévue qui pourrait survenir.
4. **IA explicable (XAI) et transparence** : Comme discuté dans la sous-section précédente, l'incorporation de techniques d'IA explicable et l'assurance de la transparence dans le processus d'auto-réparation des données peuvent aider à atténuer les conséquences imprévues. En fournissant des explications claires pour les corrections de données et en maintenant des pistes d'audit complètes, les organisations peuvent mieux comprendre et retracer le raisonnement derrière les modifications effectuées par les modèles d'IA.

5. **Approche incrémentale et itérative** : Adopter une approche incrémentale et itérative pour l'auto-réparation des données peut aider à minimiser le risque de conséquences imprévues. Au lieu d'appliquer des corrections automatisées à l'ensemble des données en une fois, commencez par un sous-ensemble de données et élargissez progressivement la portée à mesure que les techniques s'avèrent efficaces et fiables. Cela permet une surveillance et un ajustement attentifs en cours de route, réduisant l'impact de toute conséquence imprévue.
6. **Collaboration et retour d'information** : Impliquer les parties prenantes de différents domaines et encourager la collaboration et le retour d'information tout au long du processus d'auto-réparation des données peut aider à identifier et à traiter les conséquences imprévues. La recherche régulière d'contributions d'experts du domaine, de consommateurs de données et d'utilisateurs finaux peut fournir des aperçus précieux sur l'impact réel des corrections de données et mettre en évidence des problèmes qui auraient pu être négligés.

En abordant de manière proactive le risque de conséquences imprévues et en mettant en œuvre des garanties appropriées, les organisations peuvent exploiter les avantages des techniques d'auto-réparation des données tout en minimisant les effets négatifs potentiels. Il est important d'aborder l'auto-réparation des données comme un processus itératif et collaboratif, en surveillant, évaluant et affinant continuellement les techniques pour s'assurer qu'elles s'alignent sur les résultats souhaités et maintiennent l'intégrité et la fiabilité des données.

Lors de l'examen de l'utilisation des modèles d'auto-réparation des données, il est essentiel d'évaluer soigneusement ces facteurs et de peser les avantages par rapport aux risques et limitations potentiels. Dans certains cas, une approche hybride combinant des corrections automatisées avec une supervision et une intervention humaines peut être la solution la plus appropriée.

Il est également important de noter que les techniques d'auto-réparation des données ne doivent pas être considérées comme un remplacement des mécanismes robustes de validation des données, d'assainissement des entrées et de gestion des erreurs. Ces pratiques fondamentales restent essentielles pour assurer l'intégrité et la sécurité des données. L'auto-réparation des données doit être vue comme une approche complémentaire qui peut augmenter et améliorer ces mesures existantes.

En fin de compte, la décision d'employer des modèles d'auto-réparation des données dépend des exigences spécifiques, des contraintes et des priorités de votre application. En considérant attentivement les considérations décrites ci-dessus et en les alignant avec les objectifs et l'architecture de votre application, vous pouvez prendre des décisions éclairées sur quand et comment utiliser efficacement les techniques d'auto-réparation des données.

Génération de Contenu Contextuel



Les modèles de Génération de Contenu Contextuel exploitent la puissance des grands modèles de langage (GML) pour générer du contenu dynamique et contextuel au sein des applications. Cette catégorie de modèles reconnaît l'importance de fournir aux utilisateurs du contenu personnalisé et pertinent en fonction de leurs besoins spécifiques, de leurs préférences et même de leurs interactions précédentes et actuelles avec l'application.

Dans le contexte de cette approche, le “contenu” fait référence à la fois au contenu principal (c'est-à-dire les articles de blog, les articles, etc.) et au méta-contenu, comme

les recommandations relatives au contenu principal.

Les modèles de Génération de Contenu Contextuel peuvent jouer un rôle crucial dans l'amélioration de vos niveaux d'engagement utilisateur, en offrant des expériences personnalisées et en automatisant les tâches de création de contenu tant pour vous que pour vos utilisateurs. En utilisant les modèles que nous décrivons dans ce chapitre, vous pouvez créer des applications qui génèrent du contenu de manière dynamique, s'adaptant au contexte et aux entrées en temps réel.

Ces modèles fonctionnent en intégrant les GML dans les sorties de l'application, allant de l'interface utilisateur (parfois appelée "chrome"), aux emails et autres formes de notifications, ainsi qu'à toutes les chaînes de génération de contenu.

Lorsqu'un utilisateur interagit avec l'application ou déclenche une demande de contenu spécifique, l'application capture le contexte pertinent, comme les préférences de l'utilisateur, les interactions précédentes ou des invites spécifiques. Ces informations contextuelles sont ensuite transmises au GML, accompagnées de tous les modèles ou directives nécessaires, et utilisées pour produire une sortie textuelle qui aurait autrement dû être soit codée en dur, soit stockée dans une base de données, soit générée algorithmiquement.

Le contenu généré par le GML peut prendre diverses formes, comme des recommandations personnalisées, des descriptions de produits dynamiques, des réponses par email personnalisées, ou même des articles ou billets de blog entiers. L'une des utilisations les plus radicales de ce contenu que j'ai initiée il y a plus d'un an est la génération dynamique d'éléments d'interface utilisateur comme les étiquettes de formulaire, les infobulles et d'autres types de textes explicatifs.

Personnalisation

L'un des principaux avantages des modèles de Génération de Contenu Contextuel est la capacité à offrir des expériences hautement personnalisées aux utilisateurs. En générant du contenu basé sur le contexte spécifique à l'utilisateur, ces modèles permettent aux applications d'adapter le contenu aux intérêts, préférences et interactions individuels des utilisateurs.

La personnalisation va au-delà de la simple insertion du nom d'un utilisateur dans du contenu générique. Elle implique l'exploitation du riche contexte disponible sur chaque utilisateur pour générer du contenu qui résonne avec leurs besoins et désirs spécifiques. Ce contexte peut inclure un large éventail de facteurs, tels que :

1. **Informations du Profil Utilisateur** : Au niveau le plus général de l'application de cette technique, les données démographiques, les intérêts, les préférences et autres attributs du profil peuvent être utilisés pour générer du contenu qui s'aligne avec le background et les caractéristiques de l'utilisateur.
2. **Données Comportementales** : Les interactions passées d'un utilisateur avec l'application, comme les pages consultées, les liens cliqués ou les produits achetés, peuvent fournir des informations précieuses sur leur comportement et leurs intérêts. Ces données peuvent être utilisées pour générer des suggestions de contenu qui reflètent leurs schémas d'engagement et prédisent leurs besoins futurs.
3. **Facteurs Contextuels** : Le contexte actuel de l'utilisateur, comme sa localisation, son appareil, l'heure de la journée, ou même la météo, peut influencer le processus de génération de contenu. Par exemple, une application de voyage pourrait avoir un agent IA capable de générer des recommandations personnalisées basées sur la localisation actuelle de l'utilisateur et les conditions météorologiques dominantes.

En exploitant ces facteurs contextuels, les modèles de Génération de Contenu Contextuel permettent aux applications de fournir du contenu qui semble fait sur

mesure pour chaque utilisateur individuel. Ce niveau de personnalisation présente plusieurs avantages significatifs :

1. **Engagement Accru** : Le contenu personnalisé capte l'attention des utilisateurs et les maintient engagés avec l'application. Lorsque les utilisateurs sentent que le contenu est pertinent et répond directement à leurs besoins, ils sont plus susceptibles de passer plus de temps à interagir avec l'application et à explorer ses fonctionnalités.
2. **Satisfaction Utilisateur Améliorée** : Le contenu personnalisé démontre que l'application comprend et se soucie des besoins uniques de l'utilisateur. En fournissant du contenu utile, informatif et aligné sur leurs intérêts, l'application peut améliorer la satisfaction des utilisateurs et construire une connexion plus forte avec eux.
3. **Taux de Conversion Plus Élevés** : Dans le contexte des applications e-commerce ou marketing, le contenu personnalisé peut avoir un impact significatif sur les taux de conversion. En présentant aux utilisateurs des produits, des offres ou des recommandations adaptés à leurs préférences et leur comportement, l'application peut augmenter la probabilité que les utilisateurs effectuent les actions souhaitées, comme réaliser un achat ou s'inscrire à un service.

Productivité

Les modèles de Génération de Contenu Contextuel peuvent considérablement augmenter certains types de productivité en réduisant le besoin de génération et d'édition manuelle de contenu dans les processus créatifs. En exploitant la puissance des GML, vous pouvez générer du contenu de haute qualité à grande échelle, économisant le temps et l'effort que vos créateurs de contenu et développeurs devraient autrement consacrer à un travail manuel fastidieux.

Traditionnellement, les créateurs de contenu doivent rechercher, rédiger, éditer et formater le contenu pour s'assurer qu'il répond aux exigences de l'application et aux attentes des utilisateurs. Ce processus peut être chronophage et nécessiter beaucoup de ressources, particulièrement lorsque le volume de contenu augmente.

Cependant, avec les modèles de génération de contenu contextuel, le processus de création de contenu peut être largement automatisé. Les GML peuvent générer du contenu cohérent, grammaticalement correct et contextuellement pertinent en se basant sur les instructions et directives fournies. Cette automatisation offre plusieurs avantages en termes de productivité :

- 1. Réduction de l'effort manuel :** En déléguant les tâches de génération de contenu aux GML, les créateurs de contenu peuvent se concentrer sur des tâches de plus haut niveau comme la stratégie de contenu, l'idéation et l'assurance qualité. Ils peuvent fournir le contexte, les modèles et les directives nécessaires au GML et le laisser gérer la génération effective du contenu. Cela réduit l'effort manuel requis pour la rédaction et l'édition, permettant aux créateurs de contenu d'être plus productifs et efficaces.
- 2. Création de contenu plus rapide :** Les GML peuvent générer du contenu beaucoup plus rapidement que les rédacteurs humains. Avec les bonnes instructions et directives, un GML peut produire plusieurs contenus en quelques secondes ou minutes. Cette rapidité permet aux applications de générer du contenu à un rythme beaucoup plus soutenu, répondant ainsi aux demandes des utilisateurs et à l'évolution constante du paysage numérique.

La création de contenu plus rapide mène-t-elle à une "tragédie des biens communs" où l'internet se noie dans du contenu que personne ne lit? Malheureusement, je soupçonne que la réponse est oui.

3. **Cohérence et qualité** : Les GML peuvent facilement réviser le contenu pour assurer une cohérence dans le style, le ton et la qualité. Avec des directives et des exemples clairs, certains types d'applications (comme les salles de rédaction, les RP, etc.) peuvent s'assurer que leur contenu généré par l'humain s'aligne avec leur voix de marque et répond aux standards de qualité souhaités. Cette cohérence réduit le besoin de révisions et d'éditions approfondies, économisant du temps et de l'effort dans le processus de création de contenu.
4. **Itération et optimisation** : Les modèles de génération de contenu contextuel permettent une itération et une optimisation rapides du contenu. En ajustant les instructions, les modèles ou les directives fournis au GML, vos applications peuvent rapidement générer des variations de contenu et tester différentes approches de manière automatisée, ce qui était impossible auparavant. Ce processus itératif permet une expérimentation et un raffinement plus rapides des stratégies de contenu, conduisant à un contenu plus efficace et engageant au fil du temps. Cette technique particulière peut être révolutionnaire pour les applications comme le e-commerce qui dépendent fortement des taux de rebond et de l'engagement.



Il est important de noter que bien que les modèles de génération de contenu contextuel puissent grandement améliorer la productivité, ils n'éliminent pas complètement le besoin d'intervention humaine. Les créateurs et éditeurs de contenu jouent toujours un rôle crucial dans la définition de la stratégie globale de contenu, la fourniture de directives au GML et l'assurance de la qualité et de la pertinence du contenu généré.

En automatisant les aspects les plus répétitifs et chronophages de la création de contenu, les modèles de génération de contenu contextuel libèrent un temps et des ressources humaines précieux qui peuvent être réorientés vers des tâches à plus forte valeur ajoutée. Cette productivité accrue vous permet de fournir un contenu plus personnalisé et engageant aux utilisateurs tout en optimisant les flux de travail de création de contenu.

Itération et expérimentation rapides

Les modèles de génération de contenu contextuel vous permettent d'itérer rapidement et d'expérimenter différentes variations de contenu, permettant une optimisation et un raffinement plus rapides de votre stratégie de contenu. Vous pouvez générer plusieurs versions de contenu en quelques secondes, simplement en ajustant le contexte, les modèles ou les directives fournis au modèle.

Cette capacité d'itération rapide offre plusieurs avantages clés :

- 1. Tests et optimisation** : Avec la capacité de générer rapidement des variations de contenu, vous pouvez facilement tester différentes approches et mesurer leur efficacité. Par exemple, vous pouvez générer plusieurs versions d'une description de produit ou d'un message marketing, chacune adaptée à un segment d'utilisateurs ou à un contexte spécifique. En analysant les métriques d'engagement des utilisateurs, comme les taux de clics ou les taux de conversion, vous pouvez identifier les variations de contenu les plus efficaces et optimiser votre stratégie de contenu en conséquence.
- 2. Tests A/B** : Les modèles de génération de contenu contextuel permettent des tests A/B fluides du contenu. Vous pouvez générer deux ou plusieurs variations de contenu et les présenter de manière aléatoire à différents groupes d'utilisateurs. En comparant la performance de chaque variation, vous pouvez déterminer quel contenu résonne le mieux avec votre public cible. Cette approche basée sur les données vous permet de prendre des décisions éclairées et d'affiner continuellement votre contenu pour maximiser l'engagement des utilisateurs et atteindre vos objectifs souhaités.
- 3. Expériences de personnalisation** : L'itération et l'expérimentation rapides sont particulièrement précieuses en matière de personnalisation. Avec les modèles de génération de contenu contextuel, vous pouvez rapidement générer

des variations de contenu personnalisées basées sur différents segments d'utilisateurs, préférences ou comportements. En expérimentant différentes stratégies de personnalisation, vous pouvez identifier les approches les plus efficaces pour engager les utilisateurs individuels et offrir des expériences sur mesure.

4. **S'adapter aux tendances changeantes** : La capacité d'itérer et d'expérimenter rapidement vous permet de rester agile et de vous adapter aux tendances et préférences des utilisateurs en évolution. Lorsque de nouveaux sujets, mots-clés ou comportements d'utilisateurs émergent, vous pouvez rapidement générer du contenu qui s'aligne sur ces tendances. En expérimentant et en affinant continuellement votre contenu, vous pouvez rester pertinent et maintenir un avantage concurrentiel dans le paysage numérique en constante évolution.
5. **Expérimentation rentable** : L'expérimentation traditionnelle de contenu implique souvent des ressources et du temps considérables, car les créateurs de contenu doivent développer et tester manuellement différentes variations. Cependant, avec les modèles de Génération de Contenu Contextuelle, le coût de l'expérimentation est grandement réduit. Les LLMs peuvent générer des variations de contenu rapidement et à grande échelle, vous permettant d'explorer un large éventail d'idées et d'approches sans encourir de coûts substantiels.

Pour tirer le meilleur parti de l'itération rapide et de l'expérimentation, il est important d'avoir un cadre d'expérimentation bien défini en place. Ce cadre devrait inclure :

- Des objectifs clairs et des hypothèses pour chaque expérience
- Des métriques appropriées et des mécanismes de suivi pour mesurer la performance du contenu
- Des stratégies de segmentation et de ciblage pour garantir que les variations de contenu pertinentes sont servies aux bons utilisateurs
- Des outils d'analyse et de rapport pour tirer des insights des données expérimentales

- Un processus pour incorporer les apprentissages et les optimisations dans votre stratégie de contenu

En adoptant l’itération rapide et l’expérimentation, vous pouvez continuellement affiner et optimiser votre contenu, garantissant qu’il reste engageant, pertinent et efficace pour atteindre les objectifs de votre application. Cette approche agile de la création de contenu vous permet de garder une longueur d’avance et d’offrir des expériences utilisateur exceptionnelles.

Évolutivité et efficacité

À mesure que les applications se développent et que la demande de contenu personnalisé augmente, les modèles de génération de contenu contextuelle permettent une mise à l’échelle efficace de la création de contenu. Les LLMs peuvent générer du contenu pour un grand nombre d’utilisateurs et de contextes simultanément, sans nécessiter une augmentation proportionnelle des ressources humaines. Cette évolutivité permet aux applications de fournir des expériences personnalisées à une base d’utilisateurs croissante sans surcharger leurs capacités de création de contenu.



Notez que la génération de contenu contextuelle peut être utilisée efficacement pour internationaliser votre application “à la volée”. En fait, c’est exactement ce que j’ai fait en utilisant ma Gem Instant18n pour proposer Olympia dans plus d’une demi-douzaine de langues, alors que nous avons moins d’un an d’existence.

Localisation alimentée par l’IA

Si vous me permettez de me vanter un instant, je pense que ma bibliothèque Instant18n pour les applications Rails est un exemple novateur du modèle de “Génération de

Contenu Contextuelle” en action, démontrant le potentiel transformateur de l’IA dans le développement d’applications. Cette gem exploite la puissance du modèle de langage GPT d’OpenAI pour révolutionner la manière dont l’internationalisation et la localisation sont gérées dans les applications Rails.

Traditionnellement, l’internationalisation d’une application Rails implique de définir manuellement des clés de traduction et de fournir les traductions correspondantes pour chaque langue supportée. Ce processus peut être chronophage, gourmand en ressources et sujet aux incohérences. Cependant, avec la gem Instant18n, le paradigme de la localisation est complètement redéfini.

En intégrant un modèle de langage, la gem Instant18n vous permet de générer des traductions à la volée, basées sur le contexte et le sens du texte. Au lieu de s’appuyer sur des clés de traduction prédéfinies et des traductions statiques, la gem traduit dynamiquement le texte en utilisant la puissance de l’IA. Cette approche offre plusieurs avantages clés :

- 1. Localisation transparente** : Avec la gem Instant18n, les développeurs n’ont plus besoin de définir et maintenir manuellement des fichiers de traduction pour chaque langue supportée. La gem génère automatiquement des traductions basées sur le texte fourni et la langue cible souhaitée, rendant le processus de localisation sans effort et fluide.
- 2. Précision contextuelle** : L’IA peut recevoir suffisamment de contexte pour comprendre les nuances du texte à traduire. Elle peut prendre en compte le contexte environnant, les expressions idiomatiques et les références culturelles pour générer des traductions précises, naturelles et contextuellement appropriées.
- 3. Support linguistique étendu** : La gem Instant18n exploite les vastes connaissances et capacités linguistiques de GPT, permettant des traductions dans une large gamme de langues. Des langues courantes comme l’espagnol et le français aux langues plus obscures ou fictives comme le klingon et l’elfique, la gem peut gérer une grande variété de besoins en traduction.

4. **Flexibilité et créativité** : La gem va au-delà des traductions linguistiques traditionnelles et permet des options de localisation créatives et non conventionnelles. Les développeurs peuvent traduire du texte dans différents styles, dialectes, ou même des langues fictives, ouvrant de nouvelles possibilités pour des expériences utilisateur uniques et du contenu engageant.
5. **Optimisation des performances** : La gem Instant18n intègre des mécanismes de mise en cache pour améliorer les performances et réduire la surcharge des traductions répétées. Le texte traduit est mis en cache, permettant aux requêtes ultérieures pour la même traduction d'être servies rapidement sans nécessiter d'appels API redondants.

La gem Instant18n illustre la puissance du modèle de "Génération de Contenu Contextuelle" en utilisant l'IA pour générer dynamiquement du contenu localisé. Elle montre comment l'IA peut être intégrée dans la fonctionnalité centrale d'une application Rails, transformant la façon dont les développeurs abordent l'internationalisation et la localisation.

En éliminant le besoin de gérer manuellement les traductions et en permettant des traductions à la volée basées sur le contexte, la gem Instant18n fait gagner un temps et des efforts considérables aux développeurs. Elle leur permet de se concentrer sur la construction des fonctionnalités principales de leur application tout en s'assurant que l'aspect localisation est géré de manière fluide et précise.

L'Importance des Tests Utilisateur et des Retours

Enfin, gardez toujours à l'esprit l'importance des tests utilisateur et des retours. Il est crucial de valider que la génération de contenu contextuel répond aux attentes des utilisateurs et s'aligne avec les objectifs de l'application. Continuez à itérer et à affiner le contenu généré en fonction des observations des utilisateurs et des analyses.

Si vous générez du contenu dynamique à grande échelle qu'il serait impossible de valider manuellement par vous et votre équipe, envisagez d'ajouter des mécanismes de retour permettant aux utilisateurs de signaler du contenu étrange ou incorrect, accompagné d'une explication du problème. Ces précieux retours peuvent même être transmis à un agent d'IA chargé d'apporter des ajustements au composant qui a généré le contenu!

Interface utilisateur générative



L'attention est tellement précieuse de nos jours que l'engagement efficace des utilisateurs exige désormais des expériences logicielles non seulement fluides et intuitives, mais aussi hautement personnalisées selon les besoins, les préférences et les contextes individuels. En conséquence, les designers et les développeurs sont de plus en plus confrontés au défi de créer des interfaces utilisateur capables de s'adapter et de répondre aux besoins uniques de chaque utilisateur *à grande échelle*.

L'interface utilisateur générative (GenUI) est une approche véritablement révolutionnaire de la conception d'interface utilisateur qui exploite la puissance des grands modèles de langage (GML) pour créer des expériences utilisateur hautement personnalisées et dynamiques à la volée. Je voulais m'assurer de vous donner au moins une introduction à la GenUI dans ce livre, car je crois que c'est l'une des opportunités les plus prometteuses qui existe actuellement dans le domaine de la conception d'applications et des frameworks. Je suis convaincu que des dizaines de nouveaux

projets commerciaux et open-source réussis émergeront dans cette niche particulière.

À sa base, la GenUI combine les principes de la [Génération de Contenu Contextuel](#) avec des techniques d'IA avancées pour générer dynamiquement des éléments d'interface utilisateur, tels que du texte, des images et des mises en page, en se basant sur une compréhension approfondie du contexte, des préférences et des objectifs de l'utilisateur. La GenUI permet aux designers et aux développeurs de créer des interfaces qui s'adaptent et évoluent en réponse aux interactions des utilisateurs, offrant un niveau de personnalisation auparavant inatteignable.

La GenUI représente un changement fondamental dans notre approche de la conception d'interfaces utilisateur. Au lieu de concevoir pour les masses, la GenUI nous permet de concevoir pour l'individu. Le contenu et les interfaces personnalisés ont le potentiel de créer des expériences utilisateur qui résonnent plus profondément avec chaque utilisateur, augmentant l'engagement, la satisfaction et la fidélité.

En tant que technique de pointe, la transition vers la GenUI est pleine de défis conceptuels et pratiques. L'intégration de l'IA dans le processus de conception, garantir que les interfaces générées sont non seulement personnalisées mais aussi utilisables, accessibles et alignées avec l'image de marque globale et l'expérience utilisateur, tous ces défis font de la GenUI une poursuite pour quelques-uns, pas pour tous. De plus, l'implication de l'IA soulève des questions sur la confidentialité des données, la transparence et même les implications éthiques.

Malgré les défis, les expériences personnalisées à grande échelle ont le pouvoir de transformer complètement notre façon d'interagir avec les produits et services numériques. Cela ouvre des possibilités pour créer des interfaces inclusives et accessibles qui répondent aux besoins diversifiés des utilisateurs, indépendamment de leurs capacités, origines ou préférences.

Dans ce chapitre, nous explorerons le concept de GenUI, en examinant certaines caractéristiques déterminantes, les avantages clés et les défis potentiels. Nous commençons par considérer la forme la plus basique et accessible de GenUI : la

génération de texte pour des interfaces utilisateur conçues et implémentées de manière traditionnelle.

Génération de texte pour les interfaces utilisateur

Les éléments textuels qui existent dans l'interface de votre application, tels que les étiquettes de formulaire, les info-bulles et le texte explicatif, sont généralement codés en dur dans les modèles ou les composants d'interface utilisateur, offrant une expérience cohérente mais générique pour tous les utilisateurs. En utilisant des modèles de génération de contenu contextuel, vous pouvez transformer ces éléments statiques en composants dynamiques, sensibles au contexte et personnalisés.

Formulaires personnalisés

Les formulaires sont une partie omniprésente des applications web et mobiles, servant de moyen principal pour collecter les entrées des utilisateurs. Cependant, les formulaires traditionnels présentent souvent une expérience générique et impersonnelle, avec des étiquettes et des champs standard qui ne correspondent pas toujours au contexte ou aux besoins spécifiques de l'utilisateur. Les utilisateurs sont plus susceptibles de remplir des formulaires qui semblent adaptés à leurs besoins et préférences, ce qui conduit à des taux de conversion et une satisfaction utilisateur plus élevés.

Cependant, il est important de trouver un équilibre entre personnalisation et cohérence. Bien que l'adaptation des formulaires aux utilisateurs individuels puisse être bénéfique, il est crucial de maintenir un niveau de familiarité et de prévisibilité. Les utilisateurs doivent toujours pouvoir reconnaître et naviguer facilement dans les formulaires, même avec des éléments personnalisés.

Voici quelques idées de formulaires personnalisés pour l'inspiration :

Suggestions contextuelles de champs

La GenUI peut analyser les interactions précédentes de l'utilisateur, ses préférences et ses données pour fournir des suggestions intelligentes de champs comme prédictions. Par exemple, si l'utilisateur a déjà saisi son adresse de livraison, le formulaire peut automatiquement remplir les champs pertinents avec ses informations enregistrées. Cela permet non seulement de gagner du temps, mais démontre également que l'application comprend et mémorise les préférences de l'utilisateur.

Attendez une minute, cette technique ne pourrait-elle pas être réalisée sans faire appel à l'IA ? Bien sûr, mais la beauté d'implémenter ce type de fonctionnalité avec l'IA réside dans deux aspects : 1) la facilité de mise en œuvre et 2) sa résilience face aux changements et à l'évolution de votre interface utilisateur au fil du temps.

Créons rapidement un service pour notre système théorique de gestion des commandes, qui tente de remplir de manière proactive la bonne adresse de livraison pour l'utilisateur.

```
1  class OrderShippingAddressSubscriber
2      include Raix ::ChatCompletion
3
4      attr_accessor :order
5
6      delegate :customer, to : :order
7
8      DIRECTIVE = "You are a smart order processing assistant. Given the
9          customer's order history, guess the most likely shipping address
10         for the current order."
11
12     def order_created(order)
13         return unless order.pending? && order.shipping_address.blank?
14
15         self.order = order
16
17         transcript.clear
18         transcript << { system : DIRECTIVE }
19         transcript << { user : "Order History : #{order_history.to_json}" }
20         transcript << { user : "Current Order : #{order.to_json}" }
21
```

```
22     response = chat_completion
23     apply_predicted_shipping_address(order, response)
24   end
25
26   private
27
28   def apply_predicted_shipping_address(order, response)
29     # extract the shipping address from the response...
30     # ...and assume there's some sort of live update of the address fields
31     order.update(shipping_address :)
32   end
33
34   def order_history
35     customer.orders.successful.limit(100).map do |order|
36     {
37       date : order.date,
38       description : order.description,
39       shipping_address : order.shipping_address
40     }
41   end
42 end
43 end
```

Cet exemple est très simplifié, mais devrait fonctionner dans la plupart des cas. L'idée est de laisser l'IA faire une supposition de la même manière qu'un humain le ferait. Pour illustrer clairement ce dont je parle, examinons quelques données d'exemple :

```
1 Order History :
2 [
3   {"date" : "2024-01-03", "description" : "garden soil mix",
4    "shipping_address" : "123 Country Lane, Rural Town"},
5   {"date" : "2024-01-15", "description" : "hardcover fiction novels",
6    "shipping_address" : "456 City Apt, Metroville"},
7   {"date" : "2024-01-22", "description" : "baby diapers", "shipping_address" :
8    "789 Suburb St, Quietville"},
9   {"date" : "2024-02-01", "description" : "organic vegetables",
10    "shipping_address" : "123 Country Lane, Rural Town"},
11   {"date" : "2024-02-17", "description" : "mystery thriller book set",
12    "shipping_address" : "456 City Apt, Metroville"},
13   {"date" : "2024-02-25", "description" : "baby wipes",
```

```
14     "shipping_address" : "789 Suburb St, Quietville"},  
15     {"date" : "2024-03-05", "description" : "flower seeds",  
16     "shipping_address" : "123 Country Lane, Rural Town"},  
17     {"date" : "2024-03-20", "description" : "biographies",  
18     "shipping_address" : "456 City Apt, Metroville"},  
19     {"date" : "2024-03-30", "description" : "baby formula",  
20     "shipping_address" : "789 Suburb St, Quietville"},  
21     {"date" : "2024-04-12", "description" : "lawn fertilizer",  
22     "shipping_address" : "123 Country Lane, Rural Town"},  
23     {"date" : "2024-04-22", "description" : "science fiction novels",  
24     "shipping_address" : "456 City Apt, Metroville"},  
25     {"date" : "2024-05-02", "description" : "infant toys",  
26     "shipping_address" : "789 Suburb St, Quietville"},  
27     {"date" : "2024-05-14", "description" : "outdoor grill",  
28     "shipping_address" : "123 Country Lane, Rural Town"},  
29     {"date" : "2024-05-29", "description" : "literary classics",  
30     "shipping_address" : "456 City Apt, Metroville"},  
31     {"date" : "2024-06-11", "description" : "baby clothes",  
32     "shipping_address" : "789 Suburb St, Quietville"},  
33     {"date" : "2024-07-01", "description" : "watering can",  
34     "shipping_address" : "123 Country Lane, Rural Town"},  
35     {"date" : "2024-07-18", "description" : "non-fiction essays",  
36     "shipping_address" : "456 City Apt, Metroville"},  
37     {"date" : "2024-07-28", "description" : "baby bath items",  
38     "shipping_address" : "789 Suburb St, Quietville"},  
39     {"date" : "2024-08-09", "description" : "herb garden kit",  
40     "shipping_address" : "123 Country Lane, Rural Town"},  
41     {"date" : "2024-08-24", "description" : "children's books",  
42     "shipping_address" : "456 City Apt, Metroville"}  
43 ]
```

Avez-vous remarqué le motif dans les données ? Je vous garantis que c'est un jeu d'enfant pour un GML. Pour le démontrer, demandons à GPT-4 quelle est l'adresse de livraison la plus probable pour un « thermomètre ».

1 From the order history you've provided, it looks like the purchases are
2 generally clustered into three main types based on the shipping addresses :
3
4 123 Country Lane, Rural Town - This address often orders garden and
5 outdoor-related items like soil mix, vegetables, flower seeds, lawn
6 fertilizer, an outdoor grill, a watering can, and an herb garden kit.
7
8 456 City Apt, Metroville - This address tends to order various types of
9 books, indicating an interest in reading across genres like fiction,
10 thrillers, biographies, science fiction, literary classics, non-fiction
11 essays, and children's books.
12
13 789 Suburb St, Quietville - The purchases for this address are consistently
14 baby-related products, such as diapers, wipes, baby formula, infant toys,
15 baby clothes, and baby bath items.
16
17 Given these patterns, a digital thermometer could potentially be a fit for
18 any of the addresses based on different needs : outdoor temperature tracking,
19 a tool for a passionate reader interested in science gadgets, or a health
20 tool for a baby. However, considering its common use in monitoring health,
21 especially for babies, it's most likely to be purchased by someone who's
22 already buying baby care products.
23
24 Therefore, the most likely shipping address for a digital thermometer, based
25 on the purchasing patterns, would be 789 Suburb St, Quietville.

Si vous vous dites qu'il est disproportionné d'utiliser un modèle coûteux comme GPT-4 pour cette tâche, vous avez raison ! J'ai essayé le même prompt sur Mistral 7B Instruct et il a produit la réponse suivante à une vitesse de 75 jetons par seconde, pour un coût dérisoire de 0,000218 USD.

1 Based on the given order history, it appears that the customer has received
2 orders for baby-related items (diapers, wipes, formula, clothes, bath items)
3 at the address "789 Suburb St, Quietville" and garden-related items (soil,
4 seeds, watering can, herb garden kit) at the address "123 Country Lane, Rural
5 Town". The customer has also received books (fiction, non-fiction, literary
6 classics, mystery-thriller, science fiction, biographies, children's books)
7 at the address "456 City Apt, Metroville".
8
9 Given that the new order is for a "thermometer", which doesn't seem to fit
10 into any of the categories that the customer has previously ordered, it's
11 hard to accurately predict the shipping address based solely on the order
12 history. However, given the 50%-50% split between baby-related and
13 garden-related items, it could somewhat lean towards the Baby-related items
14 address ("789 Suburb St, Quietville"). But remember, this is an assumption
15 and cannot be definitively confirmed without more context or information.

Est-ce que les frais généraux et le coût de cette technique valent la peine pour rendre l'expérience d'achat plus magique? Pour de nombreux détaillants en ligne, absolument. Et d'après ce qu'on peut voir, le coût du calcul IA ne fera que diminuer, en particulier pour les fournisseurs d'hébergement de modèles open source engagés dans une course vers le bas des prix.



Utilisez un [Modèle de prompt](#) et [E/S structurées](#) avec la [Délimitation des réponses](#) pour optimiser ce type de compléTION de chat.

Ordre adaptatif des champs

L'ordre dans lequel les champs de formulaire sont présentés peut avoir un impact significatif sur l'expérience utilisateur et les taux de compléTION. Avec GenUI, vous pouvez ajuster dynamiquement l'ordre des champs en fonction du contexte de l'utilisateur et de l'importance de chaque champ. Par exemple, si l'utilisateur remplit un formulaire d'inscription pour une application de fitness, le formulaire pourrait prioriser les champs liés à ses objectifs et préférences de fitness, rendant le processus plus pertinent et engageant.

Micro-texte personnalisé

Le texte d'instruction, les messages d'erreur et autres micro-textes associés aux formulaires peuvent également être personnalisés en utilisant GenUI. Au lieu d'afficher des messages d'erreur génériques comme "Adresse e-mail invalide", vous pouvez générer des messages plus utiles et contextuels tels que "Veuillez entrer une adresse e-mail valide pour recevoir votre confirmation de commande". Ces touches personnalisées peuvent rendre l'expérience du formulaire plus conviviale et moins frustrante.

Validation personnalisée

Dans la même veine que le Micro-texte personnalisé, vous pourriez utiliser l'IA pour valider le formulaire de manière apparemment magique. Imaginez laisser une IA valider un formulaire de profil utilisateur, recherchant des erreurs potentielles au niveau *sémantique*.

Create your account

Full name

Obie Fernandez

Email

obiefenandez@gmail.com



Did you mean obiefernandez@gmail.com? [Yes, update.](#)

Country

United States



Password

.....



Nice work. This is an excellent password.

Figure 9. Pouvez-vous repérer la validation sémantique en action ?

Divulgation progressive

GenUI peut déterminer intelligemment quels champs de formulaire sont essentiels en fonction du contexte de l'utilisateur et révéler progressivement des champs supplémentaires selon les besoins. Cette technique de divulgation progressive aide à réduire la charge cognitive et rend le processus de remplissage du formulaire

plus gérable. Par exemple, si un utilisateur s'inscrit à un abonnement de base, le formulaire peut initialement ne présenter que les champs essentiels, et au fur et à mesure que l'utilisateur progresse ou sélectionne des options spécifiques, des champs supplémentaires pertinents peuvent être introduits dynamiquement.

Texte explicatif contextuel

Les info-bulles sont souvent utilisées pour fournir des informations supplémentaires ou des conseils aux utilisateurs lorsqu'ils survolent ou interagissent avec des éléments spécifiques. Avec une approche de "Génération de contenu contextuel", vous pouvez générer des info-bulles qui s'adaptent au contexte de l'utilisateur et fournissent des informations pertinentes. Par exemple, si un utilisateur explore une fonctionnalité complexe, l'info-bulle peut offrir des conseils personnalisés ou des exemples basés sur ses interactions précédentes ou son niveau de compétence.

Le texte explicatif, tel que les instructions, les descriptions ou les messages d'aide, peut être généré dynamiquement en fonction du contexte de l'utilisateur. Au lieu de présenter des explications génériques, vous pouvez utiliser les LLM pour générer du texte adapté aux besoins ou questions spécifiques de l'utilisateur. Par exemple, si un utilisateur rencontre des difficultés avec une étape particulière d'un processus, le texte explicatif peut fournir des conseils personnalisés ou des astuces de dépannage.

Le micro-texte fait référence aux petits éléments de texte qui guident les utilisateurs à travers votre application, comme les libellés de boutons, les messages d'erreur ou les invites de confirmation. En appliquant l'approche de [Génération de contenu contextuel](#) au micro-texte, vous pouvez créer une interface utilisateur adaptative qui répond aux actions de l'utilisateur et fournit du texte pertinent et utile. Par exemple, si un utilisateur est sur le point d'effectuer une action critique, l'invite de confirmation peut être générée dynamiquement pour fournir un message clair et personnalisé.

Le texte explicatif et les info-bulles personnalisés peuvent grandement améliorer le processus d'intégration des nouveaux utilisateurs. En fournissant des conseils et des

exemples spécifiques au contexte, vous pouvez aider les utilisateurs à comprendre et à naviguer rapidement dans l'application, réduisant ainsi la courbe d'apprentissage et augmentant l'adoption.

Les éléments d'interface dynamiques et contextuels peuvent également rendre l'application plus intuitive et engageante. Les utilisateurs sont plus susceptibles d'interagir avec les fonctionnalités et de les explorer lorsque le texte qui les accompagne est adapté à leurs besoins et intérêts spécifiques.

Jusqu'à présent, nous avons abordé des idées pour améliorer les paradigmes d'interface utilisateur existants avec l'IA, mais qu'en est-il de repenser de manière plus radicale la façon dont les interfaces utilisateur sont conçues et mises en œuvre ?

Définir l'interface utilisateur générative

Contrairement à la conception d'interface utilisateur traditionnelle, où les designers créent des interfaces fixes et statiques, la GenUI laisse entrevoir un avenir dans lequel nos logiciels disposent d'expériences flexibles et personnalisées qui peuvent évoluer et s'adapter en temps réel. Chaque fois que nous utilisons une interface conversationnelle basée sur l'IA, nous permettons à l'IA de s'adapter aux besoins particuliers de l'utilisateur. La GenUI va plus loin en appliquant ce niveau d'adaptabilité à l'interface *visuelle* du logiciel.

La raison pour laquelle il est possible d'expérimenter avec les idées de GenUI aujourd'hui est que les grands modèles de langage comprennent déjà la programmation et leurs connaissances de base incluent les technologies et frameworks d'interface utilisateur. La question est donc de savoir si les grands modèles de langage peuvent être utilisés pour générer des éléments d'interface utilisateur, tels que du texte, des images, des mises en page, et même des interfaces entières, qui sont adaptés à chaque utilisateur. Le modèle

pourrait être programmé pour prendre en compte divers facteurs, tels que les interactions passées de l'utilisateur, ses préférences déclarées, ses informations démographiques et le contexte d'utilisation actuel, pour créer des interfaces hautement personnalisées et pertinentes.

La GenUI diffère de la conception d'interface utilisateur traditionnelle de plusieurs manières clés :

- 1. Dynamique et adaptative** : La conception d'interface utilisateur traditionnelle implique la création d'interfaces fixes et statiques qui restent les mêmes pour tous les utilisateurs. En revanche, la GenUI permet des interfaces qui peuvent s'adapter et changer dynamiquement en fonction des besoins et du contexte de l'utilisateur. Cela signifie que la même application peut présenter différentes interfaces à différents utilisateurs ou même au même utilisateur dans différentes situations.
- 2. Personnalisation à grande échelle** : Avec la conception traditionnelle, créer des expériences personnalisées pour chaque utilisateur est souvent irréalisable en raison du temps et des ressources nécessaires. La GenUI, en revanche, permet une personnalisation à grande échelle. En exploitant l'IA, les designers peuvent créer des interfaces qui s'adaptent automatiquement aux besoins et préférences uniques de chaque utilisateur, sans avoir à concevoir et développer manuellement des interfaces distinctes pour chaque segment d'utilisateurs.
- 3. Accent sur les résultats** : La conception d'interface utilisateur traditionnelle se concentre souvent sur la création d'interfaces visuellement attrayantes et fonctionnelles. Bien que ces aspects restent importants dans la GenUI, l'accent principal se déplace vers l'atteinte des résultats souhaités par l'utilisateur. La GenUI vise à créer des interfaces optimisées pour les objectifs et les tâches spécifiques de chaque utilisateur, privilégiant l'utilisabilité et l'efficacité plutôt que des considérations purement esthétiques.
- 4. Apprentissage et amélioration continus** : Les systèmes GenUI peuvent apprendre et s'améliorer continuellement au fil du temps en fonction des

interactions et des retours des utilisateurs. Lorsque les utilisateurs interagissent avec les interfaces générées, les modèles d'IA peuvent collecter des données sur le comportement, les préférences et les résultats des utilisateurs, utilisant ces informations pour affiner et optimiser les futures générations d'interfaces. Ce processus d'apprentissage itératif permet aux systèmes GenUI de devenir de plus en plus efficaces pour répondre aux besoins des utilisateurs au fil du temps.

Il est important de noter que la GenUI n'est pas la même chose que les outils de conception assistée par IA, tels que ceux qui fournissent des suggestions ou automatisent certaines tâches de conception. Bien que ces outils puissent être utiles pour rationaliser le processus de conception, ils s'appuient toujours sur les designers pour prendre les décisions finales et créer des interfaces statiques. La GenUI, en revanche, implique que le système d'IA joue un rôle plus actif dans la génération et l'adaptation réelles des interfaces en fonction des données et du contexte de l'utilisateur.

La GenUI représente un changement significatif dans notre approche de la conception d'interface utilisateur, s'éloignant des solutions universelles au profit d'expériences hautement personnalisées et adaptatives. En exploitant la puissance de l'IA, la GenUI a le potentiel de révolutionner la façon dont nous interagissons avec les produits et services numériques, créant des interfaces plus intuitives, engageantes et efficaces pour chaque utilisateur individuel.

Exemple

Pour illustrer le concept de GenUI, considérons une application de fitness hypothétique appelée "FitAI". Cette application vise à fournir des plans d'entraînement personnalisés et des conseils nutritionnels aux utilisateurs en fonction de leurs objectifs individuels, niveaux de forme physique et préférences.

Dans une approche traditionnelle de conception d'interface utilisateur, FitAI pourrait avoir un ensemble fixe d'écrans et d'éléments qui sont les mêmes pour tous les

utilisateurs. Cependant, avec la GenUI, l'interface de l'application pourrait s'adapter dynamiquement aux besoins uniques et au contexte de chaque utilisateur.

Cette approche est un peu difficile à imaginer mettre en œuvre en 2024 et pourrait même ne pas avoir un retour sur investissement adéquat, mais elle est possible.

Voici comment cela pourrait fonctionner :

1. Intégration :

- Au lieu d'un questionnaire standard, FitAI utilise une IA conversationnelle pour recueillir des informations sur les objectifs, le niveau de forme physique actuel et les préférences de l'utilisateur.
- Sur la base de cette interaction initiale, l'IA génère une disposition de tableau de bord personnalisée, mettant en évidence les fonctionnalités et les informations les plus pertinentes pour les objectifs de l'utilisateur.
- La technologie d'IA actuelle pourrait disposer d'une sélection de composants d'écran à sa disposition pour composer le tableau de bord personnalisé.
- La technologie d'IA future pourrait assumer le rôle d'un designer d'interface utilisateur expérimenté et créer le tableau de bord *à partir de zéro*.

2. Planificateur d'entraînement :

- L'interface du planificateur d'entraînement est adaptée par l'IA pour correspondre spécifiquement au niveau d'expérience de l'utilisateur et à l'équipement disponible.
- Pour un débutant sans équipement, elle pourrait montrer des exercices simples avec le poids du corps accompagnés d'instructions détaillées et de vidéos.
- Pour un utilisateur avancé ayant accès à une salle de sport, elle pourrait afficher des routines plus complexes avec moins de contenu explicatif.

- Le contenu du planificateur d'entraînement n'est pas simplement filtré à partir d'un grand ensemble. Il peut être généré *à la volée* à partir d'une base de connaissances qui est interrogée avec un contexte incluant tout ce qui est connu sur l'utilisateur.

3. Suivi des progrès :

- L'interface de suivi des progrès évolue en fonction des objectifs de l'utilisateur et de ses schémas d'engagement.
- Si un utilisateur se concentre principalement sur la perte de poids, l'interface pourrait afficher de manière proéminente un graphique de tendance du poids et des statistiques de calories brûlées.
- Pour un utilisateur qui développe sa masse musculaire, elle pourrait mettre en évidence les gains de force et les changements de composition corporelle.
- L'IA peut adapter cette partie de l'application aux progrès réels de l'utilisateur. Si les progrès s'arrêtent pendant une période, l'application peut passer en mode où elle tente d'amener l'utilisateur à révéler les raisons de ce revers, afin de les atténuer.

4. Conseils nutritionnels :

- La section nutrition s'adapte aux préférences et restrictions alimentaires de l'utilisateur.
- Pour un utilisateur végétalien, elle pourrait montrer des suggestions de repas et des sources de protéines d'origine végétale.
- Pour un utilisateur intolérant au gluten, elle filtrerait automatiquement les aliments contenant du gluten des recommandations.
- Encore une fois, le contenu n'est pas tiré d'un immense ensemble de données de repas applicable à tous les utilisateurs, mais plutôt synthétisé à partir d'une base de connaissances contenant des informations adaptables selon la situation et les contraintes spécifiques de l'utilisateur.

- Par exemple, les recettes sont générées avec des spécifications d'ingrédients qui correspondent aux besoins caloriques en constante évolution de l'utilisateur à mesure que son niveau de forme physique et ses statistiques corporelles évoluent.

5. Éléments de motivation :

- Le contenu motivationnel et les notifications de l'application sont personnalisés en fonction du type de personnalité de l'utilisateur et de sa réponse aux différentes stratégies de motivation.
- Certains utilisateurs peuvent recevoir des messages d'encouragement, tandis que d'autres obtiennent des retours plus axés sur les données.

Dans cet exemple, GenUI permet à FitAI de créer une expérience hautement personnalisée pour chaque utilisateur, augmentant potentiellement l'engagement, la satisfaction et la probabilité d'atteindre les objectifs de fitness. Les éléments d'interface, le contenu et même la "personnalité" de l'application s'adaptent pour mieux servir les besoins et les préférences de chaque utilisateur.

Le passage à une conception orientée résultats

GenUI représente un changement fondamental dans l'approche de la conception d'interface utilisateur, passant d'une focalisation sur la création d'éléments d'interface spécifiques à une approche plus holistique, orientée résultats. Ce changement a plusieurs implications importantes :

1. Concentration sur les objectifs utilisateur :

- Les concepteurs devront réfléchir plus profondément aux objectifs des utilisateurs et aux résultats souhaités plutôt qu'aux composants d'interface spécifiques.

- L'accent sera mis sur la création de systèmes capables de générer des interfaces qui aident les utilisateurs à atteindre leurs objectifs efficacement.
- De nouveaux frameworks d'interface utilisateur émergeront pour donner aux concepteurs basés sur l'IA les outils nécessaires pour générer des expériences utilisateur *à la volée* et *à partir de zéro* plutôt que sur la base de spécifications d'écran prédéfinies.

2. Évolution du rôle des concepteurs :

- Les concepteurs passeront de la création de mises en page fixes à la définition de règles, de contraintes et de directives que les systèmes d'IA devront suivre lors de la génération d'interfaces.
- Ils devront développer des compétences dans des domaines tels que l'analyse de données, l'ingénierie des prompts et la pensée systémique pour guider efficacement les systèmes GenUI.

3. Importance de la recherche utilisateur :

- La recherche utilisateur devient encore plus critique dans un contexte GenUI, car les concepteurs doivent comprendre non seulement les préférences des utilisateurs, mais aussi comment ces préférences et besoins évoluent dans différents contextes.
- Des tests utilisateurs continus et des boucles de rétroaction seront essentiels pour affiner et améliorer la capacité de l'IA à générer des interfaces efficaces.

4. Conception pour la variabilité :

- Au lieu de créer une seule interface “parfaite”, les concepteurs devront envisager de multiples variations possibles et s'assurer que le système peut générer des interfaces appropriées pour des besoins utilisateurs divers.
- Cela inclut la conception pour les cas limites et la garantie que les interfaces générées maintiennent l'utilisabilité et l'accessibilité dans différentes configurations.

- La différenciation des produits prend de nouvelles dimensions impliquant des perspectives divergentes sur la psychologie de l'utilisateur et l'exploitation d'ensembles de données et de bases de connaissances uniques non disponibles pour les concurrents.

Défis et considérations

Bien que GenUI offre des possibilités passionnantes, elle présente également plusieurs défis et considérations :

1. Limitations techniques :

- La technologie d'IA actuelle, bien qu'avancée, a encore des limitations dans la compréhension des intentions complexes des utilisateurs et la génération d'interfaces véritablement conscientes du contexte.
- Problèmes de performance liés à la génération en temps réel d'éléments d'interface, en particulier sur les appareils moins puissants.

2. Exigences en matière de données :

- Selon le cas d'utilisation, les systèmes GenUI efficaces peuvent nécessiter des quantités importantes de données utilisateur pour générer des interfaces personnalisées.
- Les défis liés à la collecte éthique de données utilisateur authentiques soulèvent des préoccupations concernant la confidentialité et la sécurité des données, ainsi que les biais potentiels dans les données utilisées pour entraîner les modèles GenUI.

3. Utilisabilité et cohérence :

- Au moins jusqu'à ce que la pratique se généralise, une application aux interfaces en constante évolution pourrait entraîner des problèmes d'utilisabilité, car les utilisateurs pourraient avoir du mal à trouver des éléments familiers ou à naviguer efficacement.
- Il sera crucial de trouver un équilibre entre la personnalisation et le maintien d'une interface cohérente et facile à apprendre.

4. Dépendance excessive à l'IA :

- Il existe un risque de trop déléguer les décisions de conception aux systèmes d'IA, ce qui peut conduire à des choix d'interface sans inspiration, problématiques ou simplement défectueux.
- La supervision humaine et la capacité de remplacer les designs générés par l'IA resteront importantes dans un avenir prévisible.

5. Préoccupations en matière d'accessibilité :

- Garantir que les interfaces générées dynamiquement restent accessibles aux utilisateurs handicapés présente des défis entièrement nouveaux, ce qui est inquiétant étant donné le faible niveau de conformité en matière d'accessibilité démontré par les systèmes typiques.
- D'autre part, les concepteurs IA peuvent être implémentés avec une préoccupation *intégrée* pour l'accessibilité et des capacités pour construire des interfaces accessibles à la volée, tout comme ils construisent des interfaces pour les utilisateurs non handicapés.
- Dans tous les cas, les systèmes GenUI doivent être conçus avec des directives d'accessibilité robustes et des processus de test.

6. Confiance des utilisateurs et transparence :

- Les utilisateurs peuvent se sentir mal à l'aise avec des interfaces qui semblent "trop en savoir" sur eux ou qui changent d'une manière qu'ils ne comprennent pas.

- Assurer la transparence sur la façon dont et pourquoi les interfaces sont personnalisées sera important pour établir la confiance des utilisateurs.

Perspectives d'avenir et opportunités

L'avenir de l'Interface Utilisateur Générative (GenUI) est très prometteur pour révolutionner notre façon d'interagir avec les produits et services numériques. À mesure que cette technologie continue d'évoluer, nous pouvons anticiper un changement radical dans la façon dont les interfaces utilisateur sont conçues, mises en œuvre et expérimentées. Je pense que la GenUI est le phénomène qui poussera enfin nos logiciels dans le domaine de ce qui est maintenant considéré comme de la science-fiction.

L'une des perspectives les plus excitantes de la GenUI est son potentiel d'amélioration de l'accessibilité à grande échelle, allant au-delà du simple fait de s'assurer que les personnes souffrant de handicaps graves ne sont pas complètement exclues de l'utilisation de votre logiciel. En adaptant automatiquement les interfaces aux besoins individuels des utilisateurs, la GenUI pourrait rendre les expériences numériques plus inclusives que jamais. Imaginez des interfaces qui s'ajustent de manière transparente pour fournir un texte plus grand pour les utilisateurs plus jeunes ou malvoyants, ou des mises en page simplifiées pour ceux ayant des handicaps cognitifs, le tout sans nécessiter de configuration manuelle ou de versions "accessibles" séparées des applications.

Les capacités de personnalisation de la GenUI sont susceptibles d'augmenter l'engagement, la satisfaction et la fidélité des utilisateurs pour une large gamme de produits numériques. À mesure que les interfaces s'adaptent mieux aux préférences et aux comportements individuels, les utilisateurs trouveront les expériences numériques plus intuitives et agréables, ce qui pourrait conduire à des interactions plus profondes et plus significatives avec la technologie.

La GenUI a également le potentiel de transformer le processus d'intégration des nouveaux utilisateurs. En créant des expériences utilisateur intuitives et personnalisées pour les premiers contacts qui s'adaptent rapidement au niveau d'expertise de chaque utilisateur, la GenUI pourrait réduire considérablement la courbe d'apprentissage associée aux nouvelles applications. Cela pourrait conduire à des taux d'adoption plus rapides et à une confiance accrue des utilisateurs dans l'exploration de nouvelles fonctionnalités.

Une autre possibilité passionnante est la capacité de la GenUI à maintenir une expérience utilisateur cohérente sur différents appareils et plateformes tout en optimisant pour chaque contexte d'utilisation spécifique. Cela pourrait résoudre le défi de longue date consistant à fournir des expériences cohérentes dans un paysage d'appareils de plus en plus fragmenté, des smartphones et tablettes aux ordinateurs de bureau et aux technologies émergentes comme les lunettes de réalité augmentée.

La nature axée sur les données de la GenUI ouvre des opportunités pour une itération et une amélioration rapides dans la conception d'interfaces utilisateur. En recueillant des données en temps réel sur la façon dont les utilisateurs interagissent avec les interfaces générées, les designers et les développeurs peuvent obtenir des insights sans précédent sur le comportement et les préférences des utilisateurs. Cette boucle de rétroaction pourrait conduire à des améliorations continues dans la conception d'interfaces utilisateur, guidées par des modèles d'utilisation réels plutôt que par des hypothèses ou des tests utilisateur limités.

Pour se préparer à ce changement, les designers devront faire évoluer leurs compétences et leurs mentalités. L'accent passera de la création de mises en page fixes au développement de systèmes de design complets et de directives pouvant guider la génération d'interfaces par l'IA. Les designers devront cultiver une compréhension approfondie de l'analyse de données, des technologies d'IA et de la pensée systémique pour guider efficacement les systèmes GenUI.

De plus, alors que la GenUI estompe les frontières entre le design et la technologie,

les designers devront collaborer plus étroitement avec les développeurs et les data scientists. Cette approche interdisciplinaire sera cruciale pour créer des systèmes GenUI qui sont non seulement visuellement attrayants et conviviaux, mais aussi techniquement robustes et éthiquement solides.

Les implications éthiques du GenUI seront également mises au premier plan à mesure que la technologie évoluera. Les concepteurs joueront un rôle crucial dans le développement de cadres de travail pour une utilisation responsable de l'IA dans la conception d'interface, en s'assurant que la personnalisation améliore les expériences utilisateur sans compromettre la confidentialité ni manipuler le comportement des utilisateurs de manière contraire à l'éthique.

En regardant vers l'avenir, le GenUI présente à la fois des opportunités passionnantes et des défis importants. Il a le potentiel de créer des expériences numériques plus intuitives, efficaces et satisfaisantes pour les utilisateurs du monde entier. Bien qu'il exigea des concepteurs qu'ils s'adaptent et acquièrent de nouvelles compétences, il offre également une opportunité sans précédent de façonner l'avenir de l'interaction homme-machine de manière profonde et significative. Le chemin vers des systèmes GenUI pleinement réalisés sera sans doute complexe, mais les récompenses potentielles en termes d'amélioration des expériences utilisateur et d'accessibilité numérique en font un avenir pour lequel il vaut la peine de s'efforcer.

Orchestration intelligente des flux de travail



Dans le domaine du développement d'applications, les flux de travail jouent un rôle crucial dans la définition de la structure et de l'exécution des tâches, des processus et des interactions utilisateur. À mesure que les applications deviennent plus complexes et que les attentes des utilisateurs continuent d'augmenter, le besoin d'une orchestration intelligente et adaptative des flux de travail devient de plus en plus évident.

L'approche de l'"Orchestration intelligente des flux de travail" se concentre sur l'utilisation des composants d'IA pour orchestrer et optimiser dynamiquement des flux de travail complexes au sein des applications. L'objectif est de créer des applications plus efficaces, réactives et adaptables aux données et au contexte en temps réel.

Dans ce chapitre, nous explorerons les principes et modèles clés qui sous-tendent l'approche d'orchestration intelligente des flux de travail. Nous examinerons comment

l'IA peut être utilisée pour router intelligemment les tâches, automatiser la prise de décision et adapter dynamiquement les flux de travail en fonction de divers facteurs tels que le comportement des utilisateurs, les performances du système et les règles métier. À travers des exemples pratiques et des scénarios réels, nous démontrerons le potentiel transformateur de l'IA dans la rationalisation et l'optimisation des flux de travail des applications.

Que vous construisez des applications d'entreprise avec des processus métier complexes ou des applications grand public avec des parcours utilisateur dynamiques, les modèles et techniques abordés dans ce chapitre vous fourniront les connaissances et les outils nécessaires pour créer des flux de travail intelligents et efficaces qui améliorent l'expérience utilisateur globale et génèrent de la valeur commerciale.

Besoin métier

Les approches traditionnelles de la gestion des flux de travail reposent souvent sur des règles prédefinies et des arbres de décision statiques, qui peuvent être rigides, inflexibles et incapables de faire face à la nature dynamique des applications modernes.

Prenons le scénario d'une application de commerce électronique qui doit gérer un processus complexe de traitement des commandes. Le flux de travail peut impliquer plusieurs étapes telles que la validation de la commande, la vérification des stocks, le traitement des paiements, l'expédition et les notifications aux clients. Chaque étape peut avoir son propre ensemble de règles, de dépendances, d'intégrations externes et de mécanismes de gestion des exceptions. Gérer un tel flux de travail manuellement ou par le biais d'une logique codée en dur peut rapidement devenir fastidieux, sujet aux erreurs et difficile à maintenir.

De plus, à mesure que l'application évolue et que le nombre d'utilisateurs simultanés augmente, le flux de travail peut avoir besoin de s'adapter et de s'optimiser en fonction des données en temps réel et des performances du système. Par exemple, pendant les

périodes de trafic intense, l'application peut avoir besoin d'ajuster dynamiquement le flux de travail pour prioriser certaines tâches, allouer efficacement les ressources et assurer une expérience utilisateur fluide.

C'est là qu'intervient l'approche de l'"Orchestration intelligente des flux de travail". En utilisant des composants d'IA, les développeurs peuvent créer des flux de travail intelligents, adaptatifs et auto-optimisants. L'IA peut analyser de vastes quantités de données, apprendre des expériences passées et prendre des décisions éclairées en temps réel pour orchestrer efficacement le flux de travail.

Avantages clés

- Efficacité accrue** : L'IA peut optimiser l'allocation des tâches, l'utilisation des ressources et l'exécution des flux de travail, conduisant à des temps de traitement plus rapides et une meilleure efficacité globale.
- Adaptabilité** : Les flux de travail pilotés par l'IA peuvent s'adapter dynamiquement aux conditions changeantes, telles que les fluctuations de la demande des utilisateurs, les performances du système ou les exigences métier, garantissant que l'application reste réactive et résiliente.
- Prise de décision automatisée** : L'IA peut automatiser les processus complexes de prise de décision au sein du flux de travail, réduisant l'intervention manuelle et minimisant le risque d'erreurs humaines.
- Personnalisation** : L'IA peut analyser le comportement, les préférences et le contexte des utilisateurs pour personnaliser le flux de travail et offrir des expériences sur mesure à chaque utilisateur.
- Extensibilité** : Les flux de travail alimentés par l'IA peuvent s'adapter de manière transparente pour gérer des volumes croissants de données et d'interactions utilisateur, sans compromettre les performances ou la fiabilité.

Dans les sections suivantes, nous explorerons les modèles et techniques clés qui permettent la mise en œuvre de flux de travail intelligents et présenterons des exemples

concrets de la façon dont l'IA transforme la gestion des flux de travail dans les applications modernes.

Modèles clés

Pour mettre en œuvre l'orchestration intelligente des flux de travail dans les applications, les développeurs peuvent exploiter plusieurs modèles clés qui exploitent la puissance de l'IA. Ces modèles fournissent une approche structurée pour concevoir et gérer les flux de travail, permettant aux applications de s'adapter, de s'optimiser et d'automatiser les processus en fonction des données et du contexte en temps réel. Explorons certains des modèles fondamentaux de l'orchestration intelligente des flux de travail.

Routage dynamique des tâches

Ce modèle implique l'utilisation de l'IA pour router intelligemment les tâches au sein d'un flux de travail en fonction de divers facteurs tels que la priorité des tâches, la disponibilité des ressources et les performances du système. Les algorithmes d'IA peuvent analyser les caractéristiques de chaque tâche, prendre en compte l'état actuel du système et prendre des décisions éclairées pour attribuer les tâches aux ressources ou aux chemins de traitement les plus appropriés. Le routage dynamique des tâches garantit que les tâches sont distribuées et exécutées efficacement, optimisant ainsi les performances globales du flux de travail.

```
1  class TaskRouter
2    include Raix ::ChatCompletion
3    include Raix ::FunctionDispatch
4
5    attr_accessor :task
6
7    # list of functions that can be called by the AI entirely at its
8    # discretion depending on the task received
9
10   function :analyze_task_priority do
11     TaskPriorityAnalyzer.perform(task)
12   end
13
14   function :check_resource_availability, # ...
15   function :assess_system_performance, # ...
16   function :assign_task_to_resource, # ...
17
18   DIRECTIVE = "You are a task router, responsible for intelligently
19   assigning tasks to available resources based on priority, resource
20   availability, and system performance..."
21
22   def initialize(task)
23     self.task = task
24     transcript << { system : DIRECTIVE }
25     transcript << { user : task.to_json }
26   end
27
28   def perform
29     while task.unassigned?
30       chat_completion
31
32       # todo : add max loop counter and break
33     end
34
35     # capture the transcript for later analysis
36     task.update(routing_transcript : transcript)
37   end
38 end
```

Notez la boucle créée par l'expression `while` à la ligne 29, qui continue d'interroger l'IA jusqu'à ce que la tâche soit assignée. À la ligne 35, nous sauvegardons la transcription

de la tâche pour une analyse et un débogage ultérieurs si nécessaire.

Prise de Décision Contextuelle

Vous pouvez utiliser un code très similaire pour prendre des décisions contextuelles au sein d'un flux de travail. En analysant les points de données pertinents tels que les préférences utilisateur, les modèles historiques et les entrées en temps réel, les composants d'IA peuvent déterminer la ligne de conduite la plus appropriée à chaque point de décision dans le flux de travail. Adaptez le comportement de votre flux de travail en fonction du contexte spécifique de chaque utilisateur ou scénario, offrant ainsi des expériences personnalisées et optimisées.

Composition Adaptative des Flux de Travail

Ce modèle se concentre sur la composition et l'ajustement dynamiques des flux de travail en fonction de l'évolution des exigences ou des conditions. L'IA peut analyser l'état actuel du flux de travail, identifier les goulots d'étranglement ou les inefficacités, et modifier automatiquement la structure du flux de travail pour optimiser les performances. La composition adaptative des flux de travail permet aux applications d'évoluer et d'améliorer continuellement leurs processus sans nécessiter d'intervention manuelle.

Gestion et Récupération des Exceptions

La gestion et la récupération des exceptions sont des aspects critiques de l'orchestration intelligente des flux de travail. Lorsque l'on travaille avec des composants d'IA et des flux de travail complexes, il est essentiel d'anticiper et de gérer les exceptions avec élégance pour assurer la stabilité et la fiabilité du système.

Voici quelques considérations et techniques clés pour la gestion et la récupération des exceptions dans les flux de travail intelligents :

1. **Propagation des Exceptions** : Mettez en œuvre une approche cohérente pour la propagation des exceptions à travers les composants du flux de travail. Lorsqu'une exception se produit dans un composant, elle doit être capturée, enregistrée et propagée à l'orchestrateur ou à un composant distinct responsable de la gestion des exceptions. L'idée est de centraliser la gestion des exceptions et d'empêcher qu'elles ne soient silencieusement ignorées, tout en ouvrant des possibilités pour une [Gestion Intelligente des Erreurs](#).
2. **Mécanismes de Réessai** : Les mécanismes de réessai aident à améliorer la résilience du flux de travail et à gérer les échecs intermittents avec élégance. Il est certainement recommandé de mettre en œuvre des mécanismes de réessai pour les exceptions transitoires ou récupérables, comme une connectivité réseau ou une indisponibilité des ressources qui peuvent être automatiquement réessayées après un délai spécifié. Avoir un orchestrateur ou un gestionnaire d'exceptions basé sur l'IA signifie que vos stratégies de réessai n'ont pas à être mécaniques par nature, s'appuyant sur des algorithmes fixes comme le repli exponentiel. Vous pouvez laisser la gestion du réessai à la "discrétion" du composant d'IA responsable de la gestion de l'exception.
3. **Stratégies de Repli** : Si un composant d'IA ne parvient pas à fournir une réponse valide ou rencontre une erreur—une occurrence courante étant donné sa nature d'avant-garde—ayez un mécanisme de repli en place pour assurer la continuité du flux de travail. Cela peut impliquer l'utilisation de valeurs par défaut, d'algorithmes alternatifs, ou un [Humain dans la Boucle](#) pour prendre des décisions et faire avancer le flux de travail.
4. **Actions Compensatoires** : Les directives de l'orchestrateur doivent inclure des instructions sur les actions compensatoires pour gérer les exceptions qui ne peuvent pas être résolues automatiquement. Les actions compensatoires sont des étapes prises pour annuler ou atténuer les effets d'une opération échouée. Par exemple, si une étape de traitement de paiement échoue, une action compensatoire pourrait être d'annuler la transaction et d'informer l'utilisateur. Les actions

compensatoires aident à maintenir la cohérence et l'intégrité des données face aux exceptions.

5. Surveillance et Alertes des Exceptions : Mettez en place des mécanismes de surveillance et d'alerte pour détecter et notifier les parties prenantes concernées des exceptions critiques. L'orchestrateur peut être informé des seuils et des règles pour déclencher des alertes lorsque les exceptions dépassent certaines limites ou lorsque des types spécifiques d'exceptions se produisent. Cela permet une identification et une résolution proactives des problèmes avant qu'ils n'impactent l'ensemble du système.

Voici un exemple de gestion et de récupération des exceptions dans un composant de flux de travail Ruby :

```
1  class InventoryManager
2    def check_availability(order)
3      begin
4        # Perform inventory check logic
5        inventory = Inventory.find_by(product_id: order.product_id)
6        if inventory.available_quantity >= order.quantity
7          return true
8        else
9          raise InsufficientInventoryError,
10            "Insufficient inventory for product #{order.product_id}"
11        end
12      rescue InsufficientInventoryError => e
13        # Log the exception
14        logger.error("Inventory check failed : #{e.message}")
15
16        # Retry the operation after a delay
17        retry_count ||= 0
18        if retry_count < MAX_RETRIES
19          retry_count += 1
20          sleep(RETRY_DELAY)
21          retry
22        else
23          # Fallback to manual intervention
24          NotificationService.admin("Inventory check failed : Order #{order.id}")

```

```
25      return false
26  end
27 end
28 end
29 end
```

Dans cet exemple, le composant `InventoryManager` vérifie la disponibilité d'un produit pour une commande donnée. Si la quantité disponible est insuffisante, il déclenche une `InsufficientInventoryError`. L'exception est capturée, enregistrée, et un mécanisme de nouvelle tentative est mis en œuvre. Si la limite de tentatives est dépassée, le composant se replie sur une intervention manuelle en notifiant un administrateur.

En mettant en œuvre des mécanismes robustes de gestion et de récupération des exceptions, vous pouvez garantir que vos flux de travail intelligents sont résilients, maintenables et capables de gérer élégamment les situations inattendues.

Ces modèles constituent le fondement de l'orchestration des flux de travail intelligents et peuvent être combinés et adaptés pour répondre aux exigences spécifiques de différentes applications. En tirant parti de ces modèles, les développeurs peuvent créer des flux de travail qui sont flexibles, résilients et optimisés pour la performance et l'expérience utilisateur.

Dans la section suivante, nous explorerons comment ces modèles peuvent être mis en œuvre dans la pratique, en utilisant des exemples concrets et des extraits de code pour illustrer l'intégration des composants d'IA dans la gestion des flux de travail.

Mise en œuvre pratique de l'orchestration des flux de travail intelligents

Maintenant que nous avons exploré les modèles clés de l'orchestration des flux de travail intelligents, plongeons dans la façon dont ces modèles peuvent être mis en œuvre dans des applications réelles. Nous fournirons des exemples pratiques et des extraits de code pour illustrer l'intégration des composants d'IA dans la gestion des flux de travail.

Processeur de commandes intelligent

Plongeons dans un exemple pratique de mise en œuvre de l'orchestration des flux de travail intelligents en utilisant un composant OrderProcessor alimenté par l'IA dans une application e-commerce Ruby on Rails. L'OrderProcessor concrétise le concept de [Gestionnaire de processus d'intégration d'entreprise](#) que nous avons découvert au Chapitre 3 lors de la discussion sur la [Multitude de travailleurs](#). Le composant sera responsable de la gestion du flux de traitement des commandes, prenant des décisions de routage basées sur les résultats intermédiaires et orchestrant l'exécution des différentes étapes de traitement.

Le processus de traitement des commandes implique plusieurs étapes telles que la validation de la commande, la vérification des stocks, le traitement des paiements et l'expédition. Chaque étape est mise en œuvre comme un processus de travail distinct qui effectue une tâche spécifique et renvoie le résultat à l'OrderProcessor. Les étapes ne sont pas obligatoires et ne doivent pas nécessairement être effectuées dans un ordre précis.

Voici un exemple de mise en œuvre de l'OrderProcessor. Il comporte deux mixins de [Raix](#). Le premier (`ChatCompletion`) lui donne la capacité de faire de la complétion de conversation, ce qui en fait un composant d'IA. Le second (`FunctionDispatch`) permet l'appel de fonctions par l'IA, lui permettant de répondre à une invite avec une invocation de fonction plutôt qu'un message texte.

Les fonctions de travail (`validate_order`, `check_inventory`, et autres) déléguent à leurs classes de travail respectives, qui peuvent être des composants IA ou non-IA, avec pour seule exigence qu'elles renvoient les résultats de leur travail dans un format qui peut être représenté sous forme de chaîne de caractères.



Comme pour tous les autres exemples de cette partie du livre, ce code est pratiquement du pseudo-code et vise uniquement à transmettre la signification du modèle et à inspirer vos propres créations. Les descriptions complètes des modèles et les exemples de code complets sont inclus dans la Partie 2.

```
1  class OrderProcessor
2    include Raix ::ChatCompletion
3    include Raix ::FunctionDispatch
4
5    SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."
6
7    def initialize(order)
8      self.order = order
9      transcript << { system : SYSTEM_DIRECTIVE }
10     transcript << { user : order.to_json }
11   end
12
13   def perform
14     # will continue looping until `stop_looking!` is called
15     chat_completion(loop : true)
16   end
17
18   # list of functions available to be called by the AI
19   # truncated for brevity
20
21   def functions
22     [
23       {
24         name : "validate_order",
25         description : "Invoke to check validity of order",
26         parameters : {
```

```
27      ...
28  },
29  ...
30 ]
31 end
32
33 # implementation of functions that can be called by the AI
34 # entirely at its discretion, depending on the needs of the order
35
36 def validate_order
37   OrderValidationWorker.perform(@order)
38 end
39
40 def check_inventory
41   InventoryCheckWorker.perform(@order)
42 end
43
44 def process_payment
45   PaymentProcessingWorker.perform(@order)
46 end
47
48 def schedule_shipping
49   ShippingSchedulerWorker.perform(@order)
50 end
51
52 def send_confirmation
53   OrderConfirmationWorker.perform(@order)
54 end
55
56 def finished_processing
57   @order.update !(transcript :, processed_at : Time.current)
58   stop_looping !
59 end
60 end
```

Dans l'exemple, l'OrderProcessor est initialisé avec un objet de commande et maintient une transcription de l'exécution du flux de travail, dans le format de transcription de conversation typique qui est natif aux grands modèles de langage. Le contrôle complet est donné à l'IA pour orchestrer l'exécution des différentes étapes de traitement, telles que la validation de la commande, la vérification des stocks, le traitement des paiements

et l'expédition.

Chaque fois que la méthode `chat_completion` est appelée, la transcription est envoyée à l'IA pour qu'elle fournit une réponse sous forme d'appel de fonction. Il appartient entièrement à l'IA d'analyser le résultat de l'étape précédente et de déterminer l'action appropriée à prendre. Par exemple, si la vérification des stocks révèle des niveaux bas, l'`OrderProcessor` peut planifier une tâche de réapprovisionnement. Si le traitement du paiement échoue, il peut initier une nouvelle tentative ou notifier le service client.

L'exemple ci-dessus n'a pas de fonctions définies pour le réapprovisionnement ou la notification du service client, mais il pourrait absolument en avoir.

La transcription s'enrichit à chaque appel de fonction et sert d'enregistrement de l'exécution du flux de travail, incluant les résultats de chaque étape et les instructions générées par l'IA pour les étapes suivantes. Cette transcription peut être utilisée pour le débogage, l'audit et pour offrir une visibilité sur le processus de traitement des commandes.

En exploitant l'IA dans l'`OrderProcessor`, l'application e-commerce peut adapter dynamiquement le flux de travail en fonction des données en temps réel et gérer les exceptions de manière intelligente. Le composant IA peut prendre des décisions éclairées, optimiser le flux de travail et assurer un traitement fluide des commandes, même dans des scénarios complexes.

Le fait que la seule exigence pour les processus de travail soit de renvoyer une sortie intelligible que l'IA peut prendre en compte pour décider de la suite, vous fait peut-être réaliser comment cette approche peut réduire le travail de mappage entrée/sortie typiquement nécessaire lors de l'intégration de systèmes disparates entre eux.

Modérateur de Contenu Intelligent

Les applications de médias sociaux nécessitent généralement au minimum une modération de contenu pour assurer une communauté saine et sûre. Cet exemple de composant ContentModerator exploite l'IA pour orchestrer intelligemment le flux de travail de modération, prenant des décisions basées sur les caractéristiques du contenu et les résultats des différentes étapes de modération.

Le processus de modération implique plusieurs étapes telles que l'analyse de texte, la reconnaissance d'images, l'évaluation de la réputation de l'utilisateur et la révision manuelle. Chaque étape est implémentée comme un processus de travail distinct qui effectue une tâche spécifique et renvoie le résultat au ContentModerator.

Voici un exemple d'implémentation du ContentModerator :

```
1  class ContentModerator
2      include Raix ::ChatCompletion
3      include Raix ::FunctionDispatch
4
5      SYSTEM_DIRECTIVE = "You are a content moderator process manager,
6          tasked with the workflow involved in moderating user-generated content..."
7
8      def initialize(content)
9          @content = content
10         @transcript = [
11             { system : SYSTEM_DIRECTIVE },
12             { user : content.to_json }
13         ]
14     end
15
16     def perform
17         complete(@transcript)
18     end
19
20     def model
21         "openai/gpt-4"
22     end
23
```

```
24  # list of functions available to be called by the AI
25  # truncated for brevity
26
27  def functions
28  [
29  {
30      name : "analyze_text",
31      # ...
32  },
33  {
34      name : "recognize_image",
35      description : "Invoke to describe images...",
36      # ...
37  },
38  {
39      name : "assess_user_reputation",
40      # ...
41  },
42  {
43      name : "escalate_to_manual_review",
44      # ...
45  },
46  {
47      name : "approve_content",
48      # ...
49  },
50  {
51      name : "reject_content",
52      # ...
53  }
54  ]
55 end
56
57 # implementation of functions that can be called by the AI
58 # entirely at its discretion, depending on the needs of the order
59
60 def analyze_text
61     result = TextAnalysisWorker.perform(@content)
62     continue_with(result)
63 end
64
65 def recognize_image
```

```
66     result = ImageRecognitionWorker.perform(@content)
67     continue_with(result)
68   end
69
70   def assess_user_reputation
71     result = UserReputationWorker.perform(@content.user)
72     continue_with(result)
73   end
74
75   def escalate_to_manual_review
76     ManualReviewWorker.perform(@content)
77     @content.update !(status : 'pending', transcript : @transcript)
78   end
79
80   def approve_content
81     @content.update !(status : 'approved', transcript : @transcript)
82   end
83
84   def reject_content
85     @content.update !(status : 'rejected', transcript : @transcript)
86   end
87
88   private
89
90   def continue_with(result)
91     @transcript << { function : result }
92     complete(@transcript)
93   end
94 end
```

Dans cet exemple, le ContentModerator est initialisé avec un objet de contenu et maintient une transcription de modération au format conversation. Le composant IA a un contrôle total sur le flux de modération, décidant quelles étapes exécuter en fonction des caractéristiques du contenu et des résultats de chaque étape.

Les fonctions de traitement disponibles que l'IA peut invoquer comprennent `analyze_text`, `recognize_image`, `assess_user_reputation`, et `escalate_to_manual_review`. Chaque fonction délègue la tâche à un processus de traitement correspondant (`TextAnalysisWorker`, `ImageRecognitionWorker`, etc.) et ajoute le résultat à la

transcription de modération, à l'exception de la fonction d'escalade qui agit comme un état final. Enfin, les fonctions `approve_content` et `reject_content` agissent également comme des états finaux.

Le composant IA analyse le contenu et détermine l'action appropriée à prendre. Si le contenu contient des références d'images, il peut faire appel au processus `recognize-image` pour obtenir de l'aide lors d'un examen visuel. Si un processus signale un contenu potentiellement nuisible, l'IA peut décider d'escalader le contenu pour un examen manuel ou simplement le rejeter directement. Mais selon la gravité de l'avertissement, l'IA peut choisir d'utiliser les résultats de l'évaluation de la réputation de l'utilisateur pour décider comment traiter le contenu dont elle n'est pas autrement sûre. Selon le cas d'utilisation, les utilisateurs de confiance ont peut-être plus de latitude dans ce qu'ils peuvent publier. Et ainsi de suite...

Comme dans l'exemple précédent du gestionnaire de processus, la transcription de modération sert d'enregistrement de l'exécution du flux de travail, incluant les résultats de chaque étape et les décisions générées par l'IA. Cette transcription peut être utilisée pour l'audit, la transparence et l'amélioration du processus de modération au fil du temps.

En exploitant l'IA dans le `ContentModerator`, l'application de médias sociaux peut adapter dynamiquement le flux de modération en fonction des caractéristiques du contenu et gérer intelligemment des scénarios de modération complexes. Le composant IA peut prendre des décisions éclairées, optimiser le flux de travail et assurer une expérience communautaire sûre et saine.

Explorons deux autres exemples qui démontrent la planification prédictive des tâches ainsi que la gestion et la récupération des exceptions dans le contexte de l'orchestration intelligente des flux de travail.

Planification prédictive des tâches dans un système de support client

Dans une application de support client construite avec Ruby on Rails, la gestion et la priorisation efficaces des tickets d'assistance sont cruciales pour fournir une aide rapide aux clients. Le composant `SupportTicketScheduler` utilise l'IA pour planifier et attribuer de manière prédictive les tickets d'assistance aux agents disponibles en fonction de divers facteurs tels que l'urgence du ticket, l'expertise de l'agent et la charge de travail.

```
1  class SupportTicketScheduler
2    include Raix ::ChatCompletion
3    include Raix ::FunctionDispatch
4
5    SYSTEM_DIRECTIVE = "You are a support ticket scheduler,
6      tasked with intelligently assigning tickets to available agents..."
7
8    def initialize(ticket)
9      @ticket = ticket
10     @transcript = [
11       { system : SYSTEM_DIRECTIVE },
12       { user : ticket.to_json }
13     ]
14   end
15
16   def perform
17     complete(@transcript)
18   end
19
20   def model
21     "openai/gpt-4"
22   end
23
24   def functions
25     [
26       {
27         name : "analyze_ticket_urgency",
28         # ...
29       },
30       {
31         name : "assign_ticket_to_agent",
32         # ...
33       }
34     ]
35   end
36
37   def functions_for(ticket)
38     [
39       {
40         name : "analyze_ticket_urgency",
41         # ...
42       }
43     ]
44   end
45
46   def complete(transcript)
47     transcript
48   end
49
50   def self.schedule(ticket)
51     SupportTicketScheduler.new(ticket).perform
52   end
53
54   def self.model
55     "openai/gpt-4"
56   end
57
58   def self.functions
59     [
60       {
61         name : "analyze_ticket_urgency",
62         # ...
63       }
64     ]
65   end
66
67   def self.functions_for(ticket)
68     [
69       {
70         name : "analyze_ticket_urgency",
71         # ...
72       }
73     ]
74   end
75
76   def self.complete(transcript)
77     transcript
78   end
79
80   def self.schedule(ticket)
81     SupportTicketScheduler.new(ticket).perform
82   end
83
84   def self.model
85     "openai/gpt-4"
86   end
87
88   def self.functions
89     [
90       {
91         name : "analyze_ticket_urgency",
92         # ...
93       }
94     ]
95   end
96
97   def self.functions_for(ticket)
98     [
99       {
100      name : "analyze_ticket_urgency",
101      # ...
102    }
103  ]
104
105  def self.complete(transcript)
106    transcript
107  end
108
109  def self.schedule(ticket)
110    SupportTicketScheduler.new(ticket).perform
111  end
112
113  def self.model
114    "openai/gpt-4"
115  end
116
117  def self.functions
118    [
119      {
120        name : "analyze_ticket_urgency",
121        # ...
122      }
123    ]
124  end
125
126  def self.functions_for(ticket)
127    [
128      {
129        name : "analyze_ticket_urgency",
130        # ...
131      }
132    ]
133  end
134
135  def self.complete(transcript)
136    transcript
137  end
138
139  def self.schedule(ticket)
140    SupportTicketScheduler.new(ticket).perform
141  end
142
143  def self.model
144    "openai/gpt-4"
145  end
146
147  def self.functions
148    [
149      {
150        name : "analyze_ticket_urgency",
151        # ...
152      }
153    ]
154  end
155
156  def self.functions_for(ticket)
157    [
158      {
159        name : "analyze_ticket_urgency",
160        # ...
161      }
162    ]
163  end
164
165  def self.complete(transcript)
166    transcript
167  end
168
169  def self.schedule(ticket)
170    SupportTicketScheduler.new(ticket).perform
171  end
172
173  def self.model
174    "openai/gpt-4"
175  end
176
177  def self.functions
178    [
179      {
180        name : "analyze_ticket_urgency",
181        # ...
182      }
183    ]
184  end
185
186  def self.functions_for(ticket)
187    [
188      {
189        name : "analyze_ticket_urgency",
190        # ...
191      }
192    ]
193  end
194
195  def self.complete(transcript)
196    transcript
197  end
198
199  def self.schedule(ticket)
200    SupportTicketScheduler.new(ticket).perform
201  end
202
203  def self.model
204    "openai/gpt-4"
205  end
206
207  def self.functions
208    [
209      {
210        name : "analyze_ticket_urgency",
211        # ...
212      }
213    ]
214  end
215
216  def self.functions_for(ticket)
217    [
218      {
219        name : "analyze_ticket_urgency",
220        # ...
221      }
222    ]
223  end
224
225  def self.complete(transcript)
226    transcript
227  end
228
229  def self.schedule(ticket)
230    SupportTicketScheduler.new(ticket).perform
231  end
232
233  def self.model
234    "openai/gpt-4"
235  end
236
237  def self.functions
238    [
239      {
240        name : "analyze_ticket_urgency",
241        # ...
242      }
243    ]
244  end
245
246  def self.functions_for(ticket)
247    [
248      {
249        name : "analyze_ticket_urgency",
250        # ...
251      }
252    ]
253  end
254
255  def self.complete(transcript)
256    transcript
257  end
258
259  def self.schedule(ticket)
260    SupportTicketScheduler.new(ticket).perform
261  end
262
263  def self.model
264    "openai/gpt-4"
265  end
266
267  def self.functions
268    [
269      {
270        name : "analyze_ticket_urgency",
271        # ...
272      }
273    ]
274  end
275
276  def self.functions_for(ticket)
277    [
278      {
279        name : "analyze_ticket_urgency",
280        # ...
281      }
282    ]
283  end
284
285  def self.complete(transcript)
286    transcript
287  end
288
289  def self.schedule(ticket)
290    SupportTicketScheduler.new(ticket).perform
291  end
292
293  def self.model
294    "openai/gpt-4"
295  end
296
297  def self.functions
298    [
299      {
300        name : "analyze_ticket_urgency",
301        # ...
302      }
303    ]
304  end
305
306  def self.functions_for(ticket)
307    [
308      {
309        name : "analyze_ticket_urgency",
310        # ...
311      }
312    ]
313  end
314
315  def self.complete(transcript)
316    transcript
317  end
318
319  def self.schedule(ticket)
320    SupportTicketScheduler.new(ticket).perform
321  end
322
323  def self.model
324    "openai/gpt-4"
325  end
326
327  def self.functions
328    [
329      {
330        name : "analyze_ticket_urgency",
331        # ...
332      }
333    ]
334  end
335
336  def self.functions_for(ticket)
337    [
338      {
339        name : "analyze_ticket_urgency",
340        # ...
341      }
342    ]
343  end
344
345  def self.complete(transcript)
346    transcript
347  end
348
349  def self.schedule(ticket)
350    SupportTicketScheduler.new(ticket).perform
351  end
352
353  def self.model
354    "openai/gpt-4"
355  end
356
357  def self.functions
358    [
359      {
360        name : "analyze_ticket_urgency",
361        # ...
362      }
363    ]
364  end
365
366  def self.functions_for(ticket)
367    [
368      {
369        name : "analyze_ticket_urgency",
370        # ...
371      }
372    ]
373  end
374
375  def self.complete(transcript)
376    transcript
377  end
378
379  def self.schedule(ticket)
380    SupportTicketScheduler.new(ticket).perform
381  end
382
383  def self.model
384    "openai/gpt-4"
385  end
386
387  def self.functions
388    [
389      {
390        name : "analyze_ticket_urgency",
391        # ...
392      }
393    ]
394  end
395
396  def self.functions_for(ticket)
397    [
398      {
399        name : "analyze_ticket_urgency",
400        # ...
401      }
402    ]
403  end
404
405  def self.complete(transcript)
406    transcript
407  end
408
409  def self.schedule(ticket)
410    SupportTicketScheduler.new(ticket).perform
411  end
412
413  def self.model
414    "openai/gpt-4"
415  end
416
417  def self.functions
418    [
419      {
420        name : "analyze_ticket_urgency",
421        # ...
422      }
423    ]
424  end
425
426  def self.functions_for(ticket)
427    [
428      {
429        name : "analyze_ticket_urgency",
430        # ...
431      }
432    ]
433  end
434
435  def self.complete(transcript)
436    transcript
437  end
438
439  def self.schedule(ticket)
440    SupportTicketScheduler.new(ticket).perform
441  end
442
443  def self.model
444    "openai/gpt-4"
445  end
446
447  def self.functions
448    [
449      {
450        name : "analyze_ticket_urgency",
451        # ...
452      }
453    ]
454  end
455
456  def self.functions_for(ticket)
457    [
458      {
459        name : "analyze_ticket_urgency",
460        # ...
461      }
462    ]
463  end
464
465  def self.complete(transcript)
466    transcript
467  end
468
469  def self.schedule(ticket)
470    SupportTicketScheduler.new(ticket).perform
471  end
472
473  def self.model
474    "openai/gpt-4"
475  end
476
477  def self.functions
478    [
479      {
480        name : "analyze_ticket_urgency",
481        # ...
482      }
483    ]
484  end
485
486  def self.functions_for(ticket)
487    [
488      {
489        name : "analyze_ticket_urgency",
490        # ...
491      }
492    ]
493  end
494
495  def self.complete(transcript)
496    transcript
497  end
498
499  def self.schedule(ticket)
500    SupportTicketScheduler.new(ticket).perform
501  end
502
503  def self.model
504    "openai/gpt-4"
505  end
506
507  def self.functions
508    [
509      {
510        name : "analyze_ticket_urgency",
511        # ...
512      }
513    ]
514  end
515
516  def self.functions_for(ticket)
517    [
518      {
519        name : "analyze_ticket_urgency",
520        # ...
521      }
522    ]
523  end
524
525  def self.complete(transcript)
526    transcript
527  end
528
529  def self.schedule(ticket)
530    SupportTicketScheduler.new(ticket).perform
531  end
532
533  def self.model
534    "openai/gpt-4"
535  end
536
537  def self.functions
538    [
539      {
540        name : "analyze_ticket_urgency",
541        # ...
542      }
543    ]
544  end
545
546  def self.functions_for(ticket)
547    [
548      {
549        name : "analyze_ticket_urgency",
550        # ...
551      }
552    ]
553  end
554
555  def self.complete(transcript)
556    transcript
557  end
558
559  def self.schedule(ticket)
560    SupportTicketScheduler.new(ticket).perform
561  end
562
563  def self.model
564    "openai/gpt-4"
565  end
566
567  def self.functions
568    [
569      {
570        name : "analyze_ticket_urgency",
571        # ...
572      }
573    ]
574  end
575
576  def self.functions_for(ticket)
577    [
578      {
579        name : "analyze_ticket_urgency",
580        # ...
581      }
582    ]
583  end
584
585  def self.complete(transcript)
586    transcript
587  end
588
589  def self.schedule(ticket)
590    SupportTicketScheduler.new(ticket).perform
591  end
592
593  def self.model
594    "openai/gpt-4"
595  end
596
597  def self.functions
598    [
599      {
600        name : "analyze_ticket_urgency",
601        # ...
602      }
603    ]
604  end
605
606  def self.functions_for(ticket)
607    [
608      {
609        name : "analyze_ticket_urgency",
610        # ...
611      }
612    ]
613  end
614
615  def self.complete(transcript)
616    transcript
617  end
618
619  def self.schedule(ticket)
620    SupportTicketScheduler.new(ticket).perform
621  end
622
623  def self.model
624    "openai/gpt-4"
625  end
626
627  def self.functions
628    [
629      {
630        name : "analyze_ticket_urgency",
631        # ...
632      }
633    ]
634  end
635
636  def self.functions_for(ticket)
637    [
638      {
639        name : "analyze_ticket_urgency",
640        # ...
641      }
642    ]
643  end
644
645  def self.complete(transcript)
646    transcript
647  end
648
649  def self.schedule(ticket)
650    SupportTicketScheduler.new(ticket).perform
651  end
652
653  def self.model
654    "openai/gpt-4"
655  end
656
657  def self.functions
658    [
659      {
660        name : "analyze_ticket_urgency",
661        # ...
662      }
663    ]
664  end
665
666  def self.functions_for(ticket)
667    [
668      {
669        name : "analyze_ticket_urgency",
670        # ...
671      }
672    ]
673  end
674
675  def self.complete(transcript)
676    transcript
677  end
678
679  def self.schedule(ticket)
680    SupportTicketScheduler.new(ticket).perform
681  end
682
683  def self.model
684    "openai/gpt-4"
685  end
686
687  def self.functions
688    [
689      {
690        name : "analyze_ticket_urgency",
691        # ...
692      }
693    ]
694  end
695
696  def self.functions_for(ticket)
697    [
698      {
699        name : "analyze_ticket_urgency",
700        # ...
701      }
702    ]
703  end
704
705  def self.complete(transcript)
706    transcript
707  end
708
709  def self.schedule(ticket)
710    SupportTicketScheduler.new(ticket).perform
711  end
712
713  def self.model
714    "openai/gpt-4"
715  end
716
717  def self.functions
718    [
719      {
720        name : "analyze_ticket_urgency",
721        # ...
722      }
723    ]
724  end
725
726  def self.functions_for(ticket)
727    [
728      {
729        name : "analyze_ticket_urgency",
730        # ...
731      }
732    ]
733  end
734
735  def self.complete(transcript)
736    transcript
737  end
738
739  def self.schedule(ticket)
740    SupportTicketScheduler.new(ticket).perform
741  end
742
743  def self.model
744    "openai/gpt-4"
745  end
746
747  def self.functions
748    [
749      {
750        name : "analyze_ticket_urgency",
751        # ...
752      }
753    ]
754  end
755
756  def self.functions_for(ticket)
757    [
758      {
759        name : "analyze_ticket_urgency",
760        # ...
761      }
762    ]
763  end
764
765  def self.complete(transcript)
766    transcript
767  end
768
769  def self.schedule(ticket)
770    SupportTicketScheduler.new(ticket).perform
771  end
772
773  def self.model
774    "openai/gpt-4"
775  end
776
777  def self.functions
778    [
779      {
780        name : "analyze_ticket_urgency",
781        # ...
782      }
783    ]
784  end
785
786  def self.functions_for(ticket)
787    [
788      {
789        name : "analyze_ticket_urgency",
790        # ...
791      }
792    ]
793  end
794
795  def self.complete(transcript)
796    transcript
797  end
798
799  def self.schedule(ticket)
800    SupportTicketScheduler.new(ticket).perform
801  end
802
803  def self.model
804    "openai/gpt-4"
805  end
806
807  def self.functions
808    [
809      {
810        name : "analyze_ticket_urgency",
811        # ...
812      }
813    ]
814  end
815
816  def self.functions_for(ticket)
817    [
818      {
819        name : "analyze_ticket_urgency",
820        # ...
821      }
822    ]
823  end
824
825  def self.complete(transcript)
826    transcript
827  end
828
829  def self.schedule(ticket)
830    SupportTicketScheduler.new(ticket).perform
831  end
832
833  def self.model
834    "openai/gpt-4"
835  end
836
837  def self.functions
838    [
839      {
840        name : "analyze_ticket_urgency",
841        # ...
842      }
843    ]
844  end
845
846  def self.functions_for(ticket)
847    [
848      {
849        name : "analyze_ticket_urgency",
850        # ...
851      }
852    ]
853  end
854
855  def self.complete(transcript)
856    transcript
857  end
858
859  def self.schedule(ticket)
860    SupportTicketScheduler.new(ticket).perform
861  end
862
863  def self.model
864    "openai/gpt-4"
865  end
866
867  def self.functions
868    [
869      {
870        name : "analyze_ticket_urgency",
871        # ...
872      }
873    ]
874  end
875
876  def self.functions_for(ticket)
877    [
878      {
879        name : "analyze_ticket_urgency",
880        # ...
881      }
882    ]
883  end
884
885  def self.complete(transcript)
886    transcript
887  end
888
889  def self.schedule(ticket)
890    SupportTicketScheduler.new(ticket).perform
891  end
892
893  def self.model
894    "openai/gpt-4"
895  end
896
897  def self.functions
898    [
899      {
900        name : "analyze_ticket_urgency",
901        # ...
902      }
903    ]
904  end
905
906  def self.functions_for(ticket)
907    [
908      {
909        name : "analyze_ticket_urgency",
910        # ...
911      }
912    ]
913  end
914
915  def self.complete(transcript)
916    transcript
917  end
918
919  def self.schedule(ticket)
920    SupportTicketScheduler.new(ticket).perform
921  end
922
923  def self.model
924    "openai/gpt-4"
925  end
926
927  def self.functions
928    [
929      {
930        name : "analyze_ticket_urgency",
931        # ...
932      }
933    ]
934  end
935
936  def self.functions_for(ticket)
937    [
938      {
939        name : "analyze_ticket_urgency",
940        # ...
941      }
942    ]
943  end
944
945  def self.complete(transcript)
946    transcript
947  end
948
949  def self.schedule(ticket)
950    SupportTicketScheduler.new(ticket).perform
951  end
952
953  def self.model
954    "openai/gpt-4"
955  end
956
957  def self.functions
958    [
959      {
960        name : "analyze_ticket_urgency",
961        # ...
962      }
963    ]
964  end
965
966  def self.functions_for(ticket)
967    [
968      {
969        name : "analyze_ticket_urgency",
970        # ...
971      }
972    ]
973  end
974
975  def self.complete(transcript)
976    transcript
977  end
978
979  def self.schedule(ticket)
980    SupportTicketScheduler.new(ticket).perform
981  end
982
983  def self.model
984    "openai/gpt-4"
985  end
986
987  def self.functions
988    [
989      {
990        name : "analyze_ticket_urgency",
991        # ...
992      }
993    ]
994  end
995
996  def self.functions_for(ticket)
997    [
998      {
999        name : "analyze_ticket_urgency",
1000       # ...
1001     }
1002   ]
1003
1004  def self.complete(transcript)
1005    transcript
1006  end
1007
1008  def self.schedule(ticket)
1009    SupportTicketScheduler.new(ticket).perform
1010  end
1011
1012  def self.model
1013    "openai/gpt-4"
1014  end
1015
1016  def self.functions
1017    [
1018      {
1019        name : "analyze_ticket_urgency",
1020        # ...
1021      }
1022    ]
1023  end
1024
1025  def self.functions_for(ticket)
1026    [
1027      {
1028        name : "analyze_ticket_urgency",
1029        # ...
1030      }
1031    ]
1032  end
1033
1034  def self.complete(transcript)
1035    transcript
1036  end
1037
1038  def self.schedule(ticket)
1039    SupportTicketScheduler.new(ticket).perform
1040  end
1041
1042  def self.model
1043    "openai/gpt-4"
1044  end
1045
1046  def self.functions
1047    [
1048      {
1049        name : "analyze_ticket_urgency",
1050        # ...
1051      }
1052    ]
1053  end
1054
1055  def self.functions_for(ticket)
1056    [
1057      {
1058        name : "analyze_ticket_urgency",
1059        # ...
1060      }
1061    ]
1062  end
1063
1064  def self.complete(transcript)
1065    transcript
1066  end
1067
1068  def self.schedule(ticket)
1069    SupportTicketScheduler.new(ticket).perform
1070  end
1071
1072  def self.model
1073    "openai/gpt-4"
1074  end
1075
1076  def self.functions
1077    [
1078      {
1079        name : "analyze_ticket_urgency",
1080        # ...
1081      }
1082    ]
1083  end
1084
1085  def self.functions_for(ticket)
1086    [
1087      {
1088        name : "analyze_ticket_urgency",
1089        # ...
1090      }
1091    ]
1092  end
1093
1094  def self.complete(transcript)
1095    transcript
1096  end
1097
1098  def self.schedule(ticket)
1099    SupportTicketScheduler.new(ticket).perform
1100  end
1101
1102  def self.model
1103    "openai/gpt-4"
1104  end
1105
1106  def self.functions
1107    [
1108      {
1109        name : "analyze_ticket_urgency",
1110        # ...
1111      }
1112    ]
1113  end
1114
1115  def self.functions_for(ticket)
1116    [
1117      {
1118        name : "analyze_ticket_urgency",
1119        # ...
1120      }
1121    ]
1122  end
1123
1124  def self.complete(transcript)
1125    transcript
1126  end
1127
1128  def self.schedule(ticket)
1129    SupportTicketScheduler.new(ticket).perform
1130  end
1131
1132  def self.model
1133    "openai/gpt-4"
1134  end
1135
1136  def self.functions
1137    [
1138      {
1139        name : "analyze_ticket_urgency",
1140        # ...
1141      }
1142    ]
1143  end
1144
1145  def self.functions_for(ticket)
1146    [
1147      {
1148        name : "analyze_ticket_urgency",
1149        # ...
1150      }
1151    ]
1152  end
1153
1154  def self.complete(transcript)
1155    transcript
1156  end
1157
1158  def self.schedule(ticket)
1159    SupportTicketScheduler.new(ticket).perform
1160  end
1161
1162  def self.model
1163    "openai/gpt-4"
1164  end
1165
1166  def self.functions
1167    [
1168      {
1169        name : "analyze_ticket_urgency",
1170        # ...
1171      }
1172    ]
1173  end
1174
1175  def self.functions_for(ticket)
1176    [
1177      {
1178        name : "analyze_ticket_urgency",
1179        # ...
1180      }
1181    ]
1182  end
1183
1184  def self.complete(transcript)
1185    transcript
1186  end
1187
1188  def self.schedule(ticket)
1189    SupportTicketScheduler.new(ticket).perform
1190  end
1191
1192  def self.model
1193    "openai/gpt-4"
1194  end
1195
1196  def self.functions
1197    [
1198      {
1199        name : "analyze_ticket_urgency",
1200        # ...
1201      }
1202    ]
1203  end
1204
1205  def self.functions_for(ticket)
1206    [
1207      {
1208        name : "analyze_ticket_urgency",
1209        # ...
1210      }
1211    ]
1212  end
1213
1214  def self.complete(transcript)
1215    transcript
1216  end
1217
1218  def self.schedule(ticket)
1219    SupportTicketScheduler.new(ticket).perform
1220  end
1221
1222  def self.model
1223    "openai/gpt-4"
1224  end
1225
1226  def self.functions
1227    [
1228      {
1229        name : "analyze_ticket_urgency",
1230        # ...
1231      }
1232    ]
1233  end
1234
1235  def self.functions_for(ticket)
1236    [
1237      {
1238        name : "analyze_ticket_urgency",
1239        # ...
1240      }
1241    ]
1242  end
1243
1244  def self.complete(transcript)
1245    transcript
1246  end
1247
1248  def self.schedule(ticket)
1249    SupportTicketScheduler.new(ticket).perform
1250  end
1251
1252  def self.model
1253    "openai/gpt-4"
1254  end
1255
1256  def self.functions
1257    [
1258      {
1259        name : "analyze_ticket_urgency",
1260        # ...
1261      }
1262    ]
1263  end
1264
1265  def self.functions_for(ticket)
1266    [
1267      {
1268        name : "analyze_ticket_urgency",
1269        # ...
1270      }
1271    ]
1272  end
1273
1274  def self.complete(transcript)
1275    transcript
1276  end
1277
1278  def self.schedule(ticket)
1279    SupportTicketScheduler.new(ticket).perform
1280  end
1281
1282  def self.model
1283    "openai/gpt-4"
1284  end
1285
1286  def self.functions
1287    [
1288      {
1289        name : "analyze_ticket_urgency",
1290        # ...
1291      }
1292    ]
1293  end
1294
1295  def self.functions_for(ticket)
1296    [
1297      {
1298        name : "analyze_ticket_urgency",
1299        # ...
1300      }
1301    ]
1302  end
1303
1304  def self.complete(transcript)
1305    transcript
1306  end
1307
1308  def self.schedule(ticket)
1309    SupportTicketScheduler.new(ticket).perform
1310  end
1311
1312  def self.model
1313    "openai/gpt-4"
1314  end
1315
1316  def self.functions
1317    [
1318      {
1319        name : "analyze_ticket_urgency",
1320        # ...
1321      }
1322    ]
1323  end
1324
1325  def self.functions_for(ticket)
1326    [
1327      {
1328        name : "analyze_ticket_urgency",
1329        # ...
1330      }
1331    ]
1332  end
1333
1334  def self.complete(transcript)
1335    transcript
1336  end
1337
1338  def self.schedule(ticket)
1339    SupportTicketScheduler.new(ticket).perform
1340  end
1341
1342  def self.model
1343    "openai/gpt-4"
1344  end
1345
1346  def self.functions
1347    [
1348      {
1349        name : "analyze_ticket_urgency",
1350        # ...
1351      }
1352    ]
1353  end
1354
1355  def self.functions_for(ticket)
1356    [
1357      {
1358        name : "analyze_ticket_urgency",
1359        # ...
1360      }
1361    ]
1362  end
1363
1364  def self.complete(transcript)
1365    transcript
1366  end
1367
1368  def self.schedule(ticket)
1369    SupportTicketScheduler.new(ticket).perform
1370  end
1371
1372  def self.model
1373    "openai/gpt-4"
1374  end
1375
1376  def self.functions
1377    [
1378      {
1379        name : "analyze_ticket_urgency",
1380        # ...
1381      }
1382    ]
1383  end
1384
1385  def self.functions_for(ticket)
1386    [
1387      {
1388        name : "analyze_ticket_urgency",
1389        # ...
1390      }
1391    ]
1392  end
1393
1394  def self.complete(transcript)
1395    transcript
1396  end
1397
1398  def self.schedule(ticket)
1399    SupportTicketScheduler.new(ticket).perform
1400  end
1401
1402  def self.model
1403    "openai/gpt-4"
1404  end
1405
1406  def self.functions
1407    [
1408      {
1409        name : "analyze_ticket_urgency",
1410        # ...
1411      }
1412    ]
1413  end
1414
1415  def self.functions_for(ticket)
1416    [
1417      {
1418        name : "analyze_ticket_urgency",
1419        # ...
1420      }
1421    ]
1422  end
1423
1424  def self.complete(transcript)
1425    transcript
1426  end
1427
1428  def self.schedule(ticket)
1429    SupportTicketScheduler.new(ticket).perform
1430  end
1431
1432  def self.model
1433    "openai/gpt-4"
1434  end
1435
1436  def self.functions
1437    [
1438      {
1439        name : "analyze_ticket_urgency",
1440        # ...
1441      }
1442    ]
1443  end
1444
1445  def self.functions_for(ticket)
1446    [
1447      {
1448        name : "analyze_ticket_urgency",
1449        # ...
1450      }
1451    ]
1452  end
1453
1454  def self.complete(transcript)
1455    transcript
1456  end
1457
1458  def self.schedule(ticket)
1459    SupportTicketScheduler.new(ticket).perform
1460  end
1461
1462  def self.model
1463    "openai/gpt-4"
1464  end
1465
1466  def self.functions
1467    [
1468      {
1469        name : "analyze_ticket_urgency",
1470        # ...
1471      }
1472    ]
1473  end
1474
1475  def self.functions_for(ticket)
1476    [
1477      {
1478        name : "analyze_ticket_urgency",
1479        # ...
1480      }
1481    ]
1482  end
1483
1484  def self.complete(transcript)
1485    transcript
1486  end
1487
1488  def self.schedule(ticket)
1489    SupportTicketScheduler.new(ticket).perform
1490  end
1491
1492  def self.model
1493    "openai/gpt-4"
1494  end
1495
1496  def self.functions
1497    [
1498      {
1499        name : "analyze_ticket_urgency",
1500        # ...
1501      }
1502    ]
1503  end
1504
1505  def self.functions_for(ticket)
1506    [
1507      {
1508        name : "analyze_ticket_urgency",
1509        # ...
1510      }
1511    ]
1512  end
1513
1514  def self.complete(transcript)
1515    transcript
1516  end
1517
1518  def self.schedule(ticket)
1519    SupportTicketScheduler.new(ticket).perform
1520  end
1521
1522  def self.model
1523    "openai/gpt-4"
1524  end
1525
1526  def self.functions
1527    [
1528      {
1529        name : "analyze_ticket_urgency",
1530        # ...
1531      }
1532    ]
1533  end
1534
1535  def self.functions_for(ticket)
1536    [
1537      {
1538        name : "analyze_ticket_urgency",
1539        # ...
1540      }
1541    ]
1542  end
1543
1544  def self.complete(transcript)
1545    transcript
1546  end
1547
1548  def self.schedule(ticket)
1549    SupportTicketScheduler.new(ticket).perform
1550  end
1551
1552  def self.model
1553    "openai/gpt-4"
1554  end
1555
1556  def self.functions
1557    [
1558      {
1559        name : "analyze_ticket_urgency",
1560        # ...
1561      }
1562    ]
1563  end
1564
1565  def self.functions_for(ticket)
1566    [
1567      {
1568        name : "analyze_ticket_urgency",
1569        # ...
1570      }
1571    ]
1572  end
1573
1574  def self.complete(transcript)
1575    transcript
1576  end
1577
1578  def self.schedule(ticket)
1579    SupportTicketScheduler.new(ticket).perform
1580  end
1581
1582  def self.model
1583    "openai/gpt-4"
1584  end
1585
1586  def self.functions
1587    [
1588      {
1589        name : "analyze_ticket_urgency",
1590        # ...
1591      }
1592    ]
1593  end
1594
1595  def self.functions_for(ticket)
1596    [
1597      {
1598        name : "analyze_ticket_urgency",
1599        # ...
1600      }
1601    ]
1602  end
1603
1604  def self.complete(transcript)
1605    transcript
1606  end
1607
1608  def self.schedule(ticket)
1609    SupportTicketScheduler.new(ticket).perform
1610  end
1611
1612  def self.model
1613    "openai/gpt-4"
1614  end
1615
1616  def self.functions
1617    [
1618      {
1619        name : "analyze_ticket_urgency",
1620        # ...
1621      }
1622    ]
1623  end
1624
1625  def self.functions_for(ticket)
1626    [
1627      {
1628        name : "analyze_ticket_urgency",
1629        # ...
1630      }
1631    ]
1632  end
1633
1634  def self.complete(transcript)
1635    transcript
1636  end
1637
1638  def self.schedule(ticket)
1639    SupportTicketScheduler.new(ticket).perform
1640  end
1641
1642  def self.model
1643    "openai/gpt-4"
1644  end
1645
1646  def self.functions
1647    [
1648      {
1649        name : "analyze_ticket_urgency",
1650        # ...
1651      }
1652    ]
1653  end
1654
1655  def self.functions_for(ticket)
1656    [
1657      {
1658        name : "analyze_ticket_urgency",
1659        # ...
1660      }
1661    ]
1662  end
1663
1664  def self.complete(transcript)
1665    transcript
1666  end
1667
1668  def self.schedule(ticket)
1669    SupportTicketScheduler.new(ticket).perform
1670  end
1671
1672  def self.model
1673    "openai/gpt-4"
1674  end
1675
1676  def self.functions
1677    [
1678      {
1679        name : "analyze_ticket_urgency",
1680        # ...
1681      }
1682    ]
1683  end
1684
1685  def self.functions_for(ticket)
1686    [
1687      {
1688        name : "analyze_ticket_urgency",
1689        # ...
1690      }
1691    ]
1692  end
1693
1694  def self.complete(transcript)
1695    transcript
1696  end
1697
1
```

```
31      name : "list_available_agents",
32      description : "Includes expertise of available agents",
33      # ...
34    },
35    {
36      name : "predict_agent_workload",
37      description : "Uses historical data to predict upcoming workloads",
38      # ...
39    },
40    {
41      name : "assign_ticket_to_agent",
42      # ...
43    },
44    {
45      name : "reschedule_ticket",
46      # ...
47    }
48  ]
49
50
51  # implementation of functions that can be called by the AI
52  # entirely at its discretion, depending on the needs of the order
53
54  def analyze_ticket_urgency
55    result = TicketUrgencyAnalyzer.perform(@ticket)
56    continue_with(result)
57  end
58
59  def list_available_agents
60    result = ListAvailableAgents.perform
61    continue_with(result)
62  end
63
64  def predict_agent_workload
65    result = AgentWorkloadPredictor.perform
66    continue_with(result)
67  end
68
69  def assign_ticket_to_agent
70    TicketAssigner.perform(@ticket, @transcript)
71  end
72
```

```
73  def delay_assignment(until)
74      until = DateTimeStandardizer.process(until)
75      SupportTicketScheduler.delay(@ticket, @transcript, until)
76  end
77
78  private
79
80  def continue_with(result)
81      @transcript << { function : result }
82      complete(@transcript)
83  end
84 end
```

Dans cet exemple, le SupportTicketScheduler est initialisé avec un objet de ticket d’assistance et maintient une transcription de planification. Le composant IA analyse les détails du ticket et planifie de manière prédictive l’attribution du ticket en fonction de facteurs tels que l’urgence du ticket, l’expertise de l’agent et la charge de travail prévue de l’agent.

Les fonctions disponibles pour l’IA incluent analyze_ticket_urgency, list_available_agents, predict_agent_workload, et assign_ticket_to_agent. Chaque fonction délègue la tâche à un composant d’analyse ou de prédiction correspondant et ajoute le résultat à la transcription de planification. L’IA a également la possibilité de retarder l’attribution en utilisant la fonction delay_assignment.

Le composant IA examine la transcription de planification et prend des décisions éclairées sur l’attribution des tickets. Il prend en compte l’urgence du ticket, l’expertise des agents disponibles et la charge de travail prévue de chaque agent pour déterminer l’agent le plus approprié pour traiter le ticket.

En tirant parti de la planification prédictive des tâches, l’application de support client peut optimiser l’attribution des tickets, réduire les temps de réponse et améliorer la satisfaction globale des clients. La gestion proactive et efficace des tickets d’assistance garantit que les bons tickets sont attribués aux bons agents au bon moment.

Gestion des Exceptions et Récupération dans un Pipeline de Traitement de Données

La gestion des exceptions et la récupération après les échecs sont essentielles pour garantir l'intégrité des données et prévenir la perte de données. Le composant DataProcessingOrchestrator utilise l'IA pour gérer intelligemment les exceptions et orchestrer le processus de récupération dans un pipeline de traitement de données

```
1  class DataProcessingOrchestrator
2      include Raix ::ChatCompletion
3      include Raix ::FunctionDispatch
4
5      SYSTEM_DIRECTIVE = "You are a data processing orchestrator..."
6
7      def initialize(data_batch)
8          @data_batch = data_batch
9          @transcript = [
10              { system : SYSTEM_DIRECTIVE },
11              { user : data_batch.to_json }
12          ]
13      end
14
15      def perform
16          complete(@transcript)
17      end
18
19      def model
20          "openai/gpt-4"
21      end
22
23      def functions
24          [
25              {
26                  name : "validate_data",
27                  # ...
28              },
29              {
30                  name : "process_data",
31                  # ...
32              },
33          ]
34      end
```

```
33      {
34        name : "request_fix",
35        # ...
36      },
37      {
38        name : "retry_processing",
39        # ...
40      },
41      {
42        name : "mark_data_as_failed",
43        # ...
44      },
45      {
46        name : "finished",
47        # ...
48      }
49    ]
50  end
51
52  # implementation of functions that can be called by the AI
53  # entirely at its discretion, depending on the needs of the order
54
55  def validate_data
56    result = DataValidator.perform(@data_batch)
57    continue_with(result)
58  rescue ValidationException => e
59    handle_validation_exception(e)
60  end
61
62  def process_data
63    result = DataProcessor.perform(@data_batch)
64    continue_with(result)
65  rescue ProcessingException => e
66    handle_processing_exception(e)
67  end
68
69  def request_fix(description_of_fix)
70    result = SmartDataFixer.new(description_of_fix, @data_batch)
71    continue_with(result)
72  end
73
74  def retry_processing(timeout_in_seconds)
```

```
75     wait(timeout_in_seconds)
76     process_data
77   end
78
79   def mark_data_as_failed
80     @data_batch.update!(status: 'failed', transcript: @transcript)
81   end
82
83   def finished
84     @data_batch.update!(status: 'finished', transcript: @transcript)
85   end
86
87   private
88
89   def continue_with(result)
90     @transcript << { function: result }
91     complete(@transcript)
92   end
93
94   def handle_validation_exception(exception)
95     @transcript << { exception: exception.message }
96     complete(@transcript)
97   end
98
99   def handle_processing_exception(exception)
100    @transcript << { exception: exception.message }
101    complete(@transcript)
102  end
103 end
```

Dans cet exemple, le `DataProcessingOrchestrator` est initialisé avec un objet de lot de données et maintient une transcription du traitement. Le composant d'IA orchestre le pipeline de traitement des données, gérant les exceptions et se remettant des échecs selon les besoins.

Les fonctions disponibles que l'IA peut invoquer comprennent `validate_data`, `process_data`, `request_fix`, `retry_processing`, et `mark_data_as_failed`. Chaque fonction délègue la tâche à un composant de traitement de données correspondant et ajoute le résultat ou les détails de l'exception à la transcription

du traitement.

Si une exception de validation se produit pendant l'étape `validate_data`, la fonction `handle_validation_exception` ajoute les données de l'exception à la transcription et rend le contrôle à l'IA. De même, si une exception de traitement survient pendant l'étape `process_data`, l'IA peut décider de la stratégie de récupération.

Selon la nature de l'exception rencontrée, l'IA peut, à sa discrétion, décider d'appeler `request_fix`, qui délègue à un composant `SmartDataFixer` alimenté par l'IA (voir le chapitre sur les Données auto-réparatrices). Le correcteur de données reçoit une description en langage naturel de la façon dont il doit modifier le `@data_batch` pour que le traitement puisse être réessayé. Peut-être qu'une reprise réussie impliquerait de supprimer du lot de données les enregistrements qui ont échoué à la validation et/ou de les copier vers un pipeline de traitement différent pour examen humain ? Les possibilités sont presque infinies.

En intégrant la gestion et la récupération des exceptions pilotées par l'IA, l'application de traitement des données devient plus résiliente et tolérante aux pannes. Le `DataProcessingOrchestrator` gère intelligemment les exceptions, minimise la perte de données et assure l'exécution fluide du flux de travail de traitement des données.

Surveillance et journalisation

La surveillance et la journalisation fournissent une visibilité sur la progression, les performances et la santé des composants de flux de travail alimentés par l'IA, permettant aux développeurs de suivre et d'analyser le comportement du système. La mise en œuvre de mécanismes efficaces de surveillance et de journalisation est essentielle pour le débogage, l'audit et l'amélioration continue des flux de travail intelligents.

Surveillance de la progression et des performances du flux de travail

Pour assurer l'exécution fluide des flux de travail intelligents, il est important de surveiller la progression et les performances de chaque composant du flux de travail. Cela implique le suivi des métriques et des événements clés tout au long du cycle de vie du flux de travail.

Voici quelques aspects importants à surveiller :

- 1. Temps d'exécution du flux de travail** : Mesurer le temps pris par chaque composant du flux de travail pour accomplir sa tâche. Cela aide à identifier les goulets d'étranglement de performance et à optimiser l'efficacité globale du flux de travail.
- 2. Utilisation des ressources** : Surveiller l'utilisation des ressources système, telles que le CPU, la mémoire et le stockage, par chaque composant du flux de travail. Cela aide à garantir que le système fonctionne dans les limites de sa capacité et peut gérer efficacement la charge de travail.
- 3. Taux d'erreur et exceptions** : Suivre l'occurrence des erreurs et des exceptions au sein des composants du flux de travail. Cela aide à identifier les problèmes potentiels et permet une gestion proactive des erreurs et une récupération.
- 4. Points de décision et résultats** : Surveiller les points de décision au sein du flux de travail et les résultats des décisions pilotées par l'IA. Cela fournit des insights sur le comportement et l'efficacité des composants d'IA.

Les données capturées par les processus de surveillance peuvent être affichées dans des tableaux de bord ou utilisées comme entrées pour des rapports programmés qui informent les administrateurs système sur la santé du système.



Les données de surveillance peuvent être transmises à un processus d'administrateur système alimenté par l'IA pour examen et action potentielle!

Journalisation des événements et décisions clés

La journalisation est une pratique essentielle qui consiste à capturer et stocker les informations pertinentes sur les événements clés, les décisions et les exceptions qui surviennent pendant l'exécution du flux de travail.

Voici quelques aspects importants à journaliser :

- 1. Initiation et achèvement du flux de travail** : Journaliser les heures de début et de fin de chaque instance de flux de travail, ainsi que toutes les métadonnées pertinentes telles que les données d'entrée et le contexte utilisateur.
- 2. Exécution des composants** : Journaliser les détails d'exécution de chaque composant du flux de travail, y compris les paramètres d'entrée, les résultats de sortie et toutes les données intermédiaires générées.
- 3. Décisions et raisonnement de l'IA** : Journaliser les décisions prises par les composants d'IA, ainsi que le raisonnement sous-jacent ou les scores de confiance. Cela fournit de la transparence et permet l'audit des décisions pilotées par l'IA.
- 4. Exceptions et messages d'erreur** : Journaliser toutes les exceptions ou messages d'erreur rencontrés pendant l'exécution du flux de travail, y compris la trace d'appel et les informations contextuelles pertinentes.

La journalisation peut être mise en œuvre en utilisant diverses techniques, telles que l'écriture dans des fichiers journaux, le stockage des journaux dans une base de données ou l'envoi des journaux à un service de journalisation centralisé. Il est important de choisir un framework de journalisation qui offre flexibilité, évolutivité et une intégration facile avec l'architecture de l'application.

Voici un exemple de la façon dont la journalisation peut être mise en œuvre dans une application Ruby on Rails en utilisant la classe `ActiveSupport::Logger` :

```
1  class WorkflowLogger
2    def self.log(message, severity = :info)
3      @logger ||= ActiveSupport::Logger.new('workflow.log')
4      @logger.formatter ||= proc do |severity, datetime, progname, msg|
5        "#{datetime} [#{severity}] #{msg}\n"
6      end
7      @logger.send(severity, message)
8    end
9  end
10
11 # Usage example
12 WorkflowLogger.log("Workflow initiated for order ##{@order.id}")
13 WorkflowLogger.log("Payment processing completed successfully")
14 WorkflowLogger.log("Inventory check failed for item ##{@item.id}", :error)
```

En plaçant stratégiquement des énoncés de journalisation dans les composants du flux de travail et les points de décision de l'IA, les développeurs peuvent capturer des informations précieuses pour le débogage, l'audit et l'analyse.

Avantages de la surveillance et de la journalisation

La mise en œuvre de la surveillance et de la journalisation dans l'orchestration intelligente des flux de travail offre plusieurs avantages :

- 1. Débogage et dépannage** : Les journaux détaillés et les données de surveillance aident les développeurs à identifier et diagnostiquer rapidement les problèmes. Ils fournissent des informations sur le flux d'exécution du workflow, les interactions entre composants et les erreurs ou exceptions rencontrées.
- 2. Optimisation des performances** : La surveillance des métriques de performance permet aux développeurs d'identifier les goulots d'étranglement et d'optimiser les composants du flux de travail pour une meilleure efficacité. En analysant les temps d'exécution, l'utilisation des ressources et d'autres métriques, les développeurs peuvent prendre des décisions éclairées pour améliorer les performances globales du système.
- 3. Audit et conformité** : La journalisation des événements et décisions clés fournit une piste d'audit pour la conformité réglementaire et la responsabilité. Cela permet

aux organisations de suivre et de vérifier les actions prises par les composants d'IA et d'assurer le respect des règles commerciales et des exigences légales.

4. Amélioration continue : Les données de surveillance et de journalisation servent d'entrées précieuses pour l'amélioration continue des flux de travail intelligents. En analysant les données historiques, en identifiant les modèles et en mesurant l'efficacité des décisions de l'IA, les développeurs peuvent affiner et améliorer itérativement la logique d'orchestration des flux de travail.

Considérations et meilleures pratiques

Lors de la mise en œuvre de la surveillance et de la journalisation dans l'orchestration intelligente des flux de travail, considérez les meilleures pratiques suivantes :

1. Définir des métriques de surveillance claires : Identifiez les métriques et événements clés qui doivent être surveillés en fonction des exigences spécifiques du flux de travail. Concentrez-vous sur les métriques qui fournissent des informations significatives sur les performances, la santé et le comportement du système.

2. Mettre en œuvre une journalisation granulaire : Assurez-vous que les énoncés de journalisation sont placés aux points appropriés dans les composants du flux de travail et les points de décision de l'IA. Capturez les informations contextuelles pertinentes, telles que les paramètres d'entrée, les résultats de sortie et toutes les données intermédiaires générées.

3. Utiliser une journalisation structurée : Adoptez un format de journalisation structuré pour faciliter l'analyse et le traitement des données de journal. La journalisation structurée permet une meilleure recherche, filtrage et agrégation des entrées de journal.

4. Gérer la conservation et la rotation des journaux : Mettez en œuvre des politiques de conservation et de rotation des journaux pour gérer le stockage et le cycle de vie des fichiers journaux. Déterminez la période de conservation appropriée en fonction

des exigences légales, des contraintes de stockage et des besoins d'analyse. Si possible, déléguez la journalisation à un service tiers tel que [Papertrail](#).

5. Sécuriser les informations sensibles : Soyez prudent lors de la journalisation d'informations sensibles, comme les informations personnelles identifiables (PII) ou les données commerciales confidentielles. Mettez en œuvre des mesures de sécurité appropriées, telles que le masquage des données ou le chiffrement, pour protéger les informations sensibles dans les fichiers journaux.

6. Intégrer des outils de surveillance et d'alerte : Utilisez des outils de surveillance et d'alerte pour centraliser la collecte, l'analyse et la visualisation des données de surveillance et de journalisation. Ces outils peuvent fournir des informations en temps réel, générer des alertes basées sur des seuils prédéfinis et faciliter la détection et la résolution proactive des problèmes. Mon outil préféré parmi ceux-ci est [Datadog](#).

En mettant en œuvre des mécanismes complets de surveillance et de journalisation, les développeurs peuvent obtenir des informations précieuses sur le comportement et les performances des flux de travail intelligents. Ces informations permettent un débogage efficace, une optimisation et une amélioration continue des systèmes d'orchestration des flux de travail alimentés par l'IA.

Considérations d'extensibilité et de performance

L'extensibilité et la performance sont des aspects critiques à prendre en compte lors de la conception et de la mise en œuvre de systèmes d'orchestration de flux de travail intelligents. À mesure que le volume de flux de travail simultanés et la complexité des composants alimentés par l'IA augmentent, il devient essentiel de s'assurer que le système peut gérer la charge de travail efficacement et s'adapter de manière transparente pour répondre aux demandes croissantes.

Gestion de volumes élevés de flux de travail simultanés

Les systèmes d'orchestration de flux de travail intelligents doivent souvent gérer un grand nombre de flux de travail simultanés. Pour assurer l'extensibilité, considérez les stratégies suivantes :

- 1. Traitement asynchrone** : Mettez en œuvre des mécanismes de traitement asynchrone pour découpler l'exécution des composants du flux de travail. Cela permet au système de gérer plusieurs flux de travail simultanément sans bloquer ni attendre que chaque composant se termine. Le traitement asynchrone peut être réalisé à l'aide de files d'attente de messages, d'architectures événementielles ou de frameworks de traitement de tâches en arrière-plan comme Sidekiq.
- 2. Architecture distribuée** : Concevez l'architecture du système pour utiliser des composants sans serveur (comme AWS Lambda) ou simplement distribuer la charge de travail sur plusieurs nœuds ou serveurs aux côtés de votre serveur d'application principal. Cela permet une extensibilité horizontale, où des nœuds supplémentaires peuvent être ajoutés pour gérer l'augmentation des volumes de flux de travail.
- 3. Exécution parallèle** : Identifiez les opportunités d'exécution parallèle au sein des flux de travail. Certains composants du flux de travail peuvent être indépendants les uns des autres et peuvent être exécutés simultanément. En utilisant des techniques de traitement parallèle, telles que le multithreading ou les files d'attente de tâches distribuées, le système peut optimiser l'utilisation des ressources et réduire le temps d'exécution global du flux de travail.

Optimisation des performances des composants basés sur l'IA

Les composants basés sur l'IA, tels que les modèles d'apprentissage automatique ou les moteurs de traitement du langage naturel, peuvent être gourmands en ressources de calcul et impacter les performances globales du système d'orchestration des flux de

travail. Pour optimiser les performances des composants d'IA, considérez les techniques suivantes :

- 1. Mise en cache** : Si votre traitement IA est purement génératif et n'implique pas de recherches d'informations en temps réel ou d'intégrations externes pour générer ses réponses, vous pouvez alors explorer les mécanismes de mise en cache pour stocker et réutiliser les résultats des opérations fréquemment accédées ou coûteuses en calcul.
- 2. Optimisation du modèle** : Optimisez continuellement la façon dont vous utilisez les modèles d'IA dans les composants du flux de travail. Cela peut impliquer des techniques telles que la *Distillation de prompts* ou simplement consister à tester de nouveaux modèles au fur et à mesure qu'ils deviennent disponibles.
- 3. Traitement par lots** : Si vous travaillez avec des modèles de classe GPT-4, vous pourriez être en mesure d'utiliser des techniques de traitement par lots pour traiter plusieurs points de données ou requêtes en une seule fois, plutôt que de les traiter individuellement. En traitant les données par lots, le système peut optimiser l'utilisation des ressources et réduire la surcharge des requêtes répétées au modèle.

Surveillance et profilage des performances

Pour identifier les goulets d'étranglement de performance et optimiser l'évolutivité du système d'orchestration intelligent des flux de travail, il est crucial de mettre en place des mécanismes de surveillance et de profilage. Considérez les approches suivantes :

- 1. Métriques de performance** : Définissez et suivez les métriques de performance clés, telles que le temps de réponse, le débit, l'utilisation des ressources et la latence. Ces métriques fournissent des informations sur les performances du système et aident à identifier les domaines à optimiser. L'agrégateur de modèles d'IA populaire [OpenRouter](#) inclut les métriques Host¹ et Speed² dans chaque réponse API, rendant trivial le suivi de

¹Host représente le temps nécessaire pour recevoir le premier octet de la génération en streaming depuis l'hôte du modèle, aussi connu sous le nom de "temps jusqu'au premier octet."

²Speed est calculée comme le nombre de tokens de complétion divisé par le temps total de génération. Pour les requêtes non streamées, la latence est considérée comme faisant partie du temps de génération.

ces métriques clés.

2. **Outils de profilage** : Utilisez des outils de profilage pour analyser les performances des composants individuels du flux de travail et des opérations d'IA. Les outils de profilage peuvent aider à identifier les points chauds de performance, les chemins de code inefficaces ou les opérations gourmandes en ressources. Les outils de profilage populaires incluent New Relic, Scout, ou les profileurs intégrés fournis par le langage de programmation ou le framework.
3. **Tests de charge** : Effectuez des tests de charge pour évaluer les performances du système sous différents niveaux de charge de travail simultanée. Les tests de charge aident à identifier les limites d'évolutivité du système, à détecter la dégradation des performances et à s'assurer que le système peut gérer le trafic attendu sans compromettre les performances.
4. **Surveillance continue** : Mettez en place des mécanismes de surveillance continue et d'alerte pour détecter de manière proactive les problèmes de performance et les goulets d'étranglement. Configurez des tableaux de bord de surveillance et des alertes pour suivre les indicateurs clés de performance (ICP) et recevoir des notifications lorsque les seuils prédéfinis sont dépassés. Cela permet une identification et une résolution rapides des problèmes de performance.

Stratégies de mise à l'échelle

Pour gérer l'augmentation des charges de travail et assurer l'évolutivité du système d'orchestration intelligent des flux de travail, considérez les stratégies de mise à l'échelle suivantes :

1. **Mise à l'échelle verticale** : La mise à l'échelle verticale consiste à augmenter les ressources (par exemple, CPU, mémoire) des nœuds ou serveurs individuels pour gérer des charges de travail plus importantes. Cette approche est appropriée lorsque le système nécessite plus de puissance de traitement ou de mémoire pour gérer des flux de travail complexes ou des opérations d'IA.

2. Mise à l'échelle horizontale : La mise à l'échelle horizontale consiste à ajouter plus de nœuds ou de serveurs au système pour distribuer la charge de travail. Cette approche est efficace lorsque le système doit gérer un grand nombre de flux de travail simultanés ou lorsque la charge de travail peut être facilement distribuée sur plusieurs nœuds. La mise à l'échelle horizontale nécessite une architecture distribuée et des mécanismes d'équilibrage de charge pour assurer une distribution uniforme du trafic.

3. Mise à l'échelle automatique : Implémentez des mécanismes de mise à l'échelle automatique pour ajuster automatiquement le nombre de nœuds ou de ressources en fonction de la demande de charge de travail. La mise à l'échelle automatique permet au système de s'adapter dynamiquement à la hausse ou à la baisse en fonction du trafic entrant, assurant une utilisation optimale des ressources et une rentabilité. Les plateformes cloud comme Amazon Web Services (AWS) ou Google Cloud Platform (GCP) fournissent des capacités de mise à l'échelle automatique qui peuvent être exploitées pour les systèmes d'orchestration intelligente des flux de travail.

Techniques d'optimisation des performances

En plus des stratégies de mise à l'échelle, considérez les techniques d'optimisation des performances suivantes pour améliorer l'efficacité du système d'orchestration intelligent des flux de travail :

1. Stockage et récupération efficaces des données : Optimisez les mécanismes de stockage et de récupération des données utilisés par les composants du flux de travail. Utilisez l'indexation efficace des bases de données, les techniques d'optimisation des requêtes et la mise en cache des données pour minimiser la latence et améliorer les performances des opérations intensives en données.

2. E/S asynchrone : Utilisez les opérations d'E/S asynchrones pour éviter les blocages et améliorer la réactivité du système. L'E/S asynchrone permet au système de gérer plusieurs requêtes simultanément sans attendre la fin des opérations d'E/S, maximisant ainsi l'utilisation des ressources.

3. Sérialisation et désérialisation efficaces : Optimisez les processus de sérialisation et de désérialisation utilisés pour l'échange de données entre les composants du flux de travail. Utilisez des formats de sérialisation efficaces, tels que Protocol Buffers ou MessagePack, pour réduire la surcharge de la sérialisation des données et améliorer les performances de la communication entre les composants.



Pour les applications basées sur Ruby, envisagez d'utiliser [Universal ID](#).

Universal ID exploite à la fois MessagePack et Brotli (une combinaison conçue pour la vitesse et la compression de données de premier ordre).

Ensemble, ces bibliothèques sont jusqu'à 30 % plus rapides et atteignent des taux de compression à 2-5 % près par rapport à Protocol Buffers.

4. Compression et encodage : Appliquez des techniques de compression et d'encodage pour réduire la taille des données transférées entre les composants du flux de travail. Les algorithmes de compression, tels que gzip ou Brotli, peuvent réduire considérablement l'utilisation de la bande passante réseau et améliorer les performances globales du système.

En prenant en compte les aspects de scalabilité et de performance lors de la conception et de l'implémentation des systèmes d'orchestration de flux de travail intelligents, vous pouvez garantir que votre système peut gérer des volumes élevés de flux de travail simultanés, optimiser les performances des composants alimentés par l'IA et évoluer de manière transparente pour répondre aux demandes croissantes. La surveillance continue, le profilage et les efforts d'optimisation sont essentiels pour maintenir les performances et la réactivité du système à mesure que la charge de travail et la complexité augmentent au fil du temps.

Tests et validation des flux de travail

Les tests et la validation sont des aspects critiques du développement et de la maintenance des systèmes d'orchestration de flux de travail intelligents. Étant donné la

nature complexe des flux de travail alimentés par l'IA, il est essentiel de s'assurer que chaque composant fonctionne comme prévu, que le flux de travail global se comporte correctement et que les décisions de l'IA sont précises et fiables. Dans cette section, nous explorerons diverses techniques et considérations pour tester et valider les flux de travail intelligents.

Tests unitaires des composants du flux de travail

Les tests unitaires consistent à tester individuellement les composants du flux de travail pour vérifier leur exactitude et leur robustesse. Lors des tests unitaires des composants de flux de travail alimentés par l'IA, considérez les points suivants :

- 1. Validation des entrées** : Testez la capacité du composant à gérer différents types d'entrées, y compris les données valides et invalides. Vérifiez que le composant gère correctement les cas limites et fournit des messages d'erreur ou des exceptions appropriés.
- 2. Vérification des sorties** : Assurez-vous que le composant produit la sortie attendue pour un ensemble donné d'entrées. Comparez la sortie réelle avec les résultats attendus pour garantir l'exactitude.
- 3. Gestion des erreurs** : Testez les mécanismes de gestion des erreurs du composant en simulant divers scénarios d'erreur, tels que des entrées invalides, l'indisponibilité des ressources ou des exceptions inattendues. Vérifiez que le composant capture et gère les erreurs de manière appropriée.
- 4. Conditions limites** : Testez le comportement du composant dans des conditions limites, telles que les entrées vides, la taille maximale d'entrée ou les valeurs extrêmes. Assurez-vous que le composant gère ces conditions avec élégance sans plantage ni production de résultats incorrects.

Voici un exemple de test unitaire pour un composant de flux de travail en Ruby utilisant le framework de test RSpec :

```
1 RSpec.describe OrderValidator do
2   describe '#validate' do
3     context 'when order is valid' do
4       let( :order) { build( :order) }
5
6       it 'returns true' do
7         expect(subject.validate(order)).to be true
8       end
9     end
10
11    context 'when order is invalid' do
12      let( :order) { build( :order, total_amount : -100) }
13
14      it 'returns false' do
15        expect(subject.validate(order)).to be false
16      end
17    end
18  end
19 end
```

Dans cet exemple, le composant `OrderValidator` est testé à l'aide de deux cas de test : un pour une commande valide et un autre pour une commande invalide. Les cas de test vérifient que la méthode `validate` renvoie la valeur booléenne attendue en fonction de la validité de la commande.

Tests d'intégration des interactions du workflow

Les tests d'intégration se concentrent sur la vérification des interactions et du flux de données entre les différents composants du workflow. Ils garantissent que les composants fonctionnent ensemble de manière transparente et produisent les résultats attendus. Lors des tests d'intégration des workflows intelligents, il faut prendre en compte les points suivants :

1. Interaction des composants : Tester la communication et l'échange de données entre les composants du workflow. Vérifier que la sortie d'un composant est correctement transmise comme entrée au composant suivant dans le workflow.

2. Cohérence des données : S'assurer que les données restent cohérentes et précises tout au long du workflow. Vérifier que les transformations de données, les calculs et les agrégations sont effectués correctement.

3. Propagation des exceptions : Tester comment les exceptions et les erreurs sont propagées et gérées à travers les composants du workflow. Vérifier que les exceptions sont capturées, enregistrées et traitées de manière appropriée pour éviter toute perturbation du workflow.

4. Comportement asynchrone : Si le workflow implique des composants asynchrones ou une exécution parallèle, tester les mécanismes de coordination et de synchronisation. S'assurer que le workflow se comporte correctement dans des scénarios concurrents et asynchrones.

Voici un exemple de test d'intégration pour un workflow en Ruby utilisant le framework de test RSpec :

```
1 RSpec.describe OrderProcessingWorkflow do
2
3   let( :order) { build( :order) }
4
5   it 'processes the order successfully' do
6     expect(OrderValidator).to receive( :validate).and_return(true)
7     expect(InventoryManager).to receive( :check_availability).and_return(true)
8     expect(PaymentProcessor).to receive( :process_payment).and_return(true)
9     expect(ShippingService).to receive( :schedule_shipping).and_return(true)
10
11   workflow = OrderProcessingWorkflow.new(order)
12   result = workflow.process
13
14   expect(result).to be true
15   expect(order.status).to eq('processed')
16 end
17
18 end
```

Dans cet exemple, le OrderProcessingWorkflow est testé en vérifiant les interactions entre les différents composants du workflow. Le cas de test établit des attentes pour le

comportement de chaque composant et s'assure que le workflow traite la commande avec succès, en mettant à jour l'état de la commande en conséquence.

Test des points de décision de l'IA

Tester les points de décision de l'IA est crucial pour garantir l'exactitude et la fiabilité des workflows basés sur l'IA. Lors du test des points de décision de l'IA, considérez les points suivants :

- 1. Précision des décisions** : Vérifiez que le composant d'IA prend des décisions précises basées sur les données d'entrée et le modèle entraîné. Comparez les décisions de l'IA avec les résultats attendus ou les données de référence.
- 2. Cas limites** : Testez le comportement du composant d'IA dans des cas limites et des scénarios inhabituels. Vérifiez que le composant d'IA gère ces cas avec élégance et prend des décisions raisonnables.
- 3. Biais et équité** : Évaluez le composant d'IA pour détecter d'éventuels biais et assurez-vous qu'il prend des décisions justes et impartiales. Testez le composant avec des données d'entrée diverses et analysez les résultats pour identifier d'éventuels schémas discriminatoires.
- 4. Explicabilité** : Si le composant d'IA fournit des explications ou un raisonnement pour ses décisions, vérifiez l'exactitude et la clarté des explications. Assurez-vous que les explications correspondent au processus de prise de décision sous-jacent.

Voici un exemple de test d'un point de décision d'IA en Ruby utilisant le framework de test RSpec :

```
1 RSpec.describe FraudDetector do
2   describe '#detect_fraud' do
3     context 'when transaction is fraudulent' do
4       let( :tx) do
5         build( :transaction, amount : 10_000, location : 'High-Risk Country')
6       end
7
8       it 'returns true' do
9         expect(subject.detect_fraud(tx)).to be true
10      end
11    end
12
13    context 'when transaction is legitimate' do
14      let( :tx) do
15        build( :transaction, amount : 100, location : 'Low-Risk Country')
16      end
17
18      it 'returns false' do
19        expect(subject.detect_fraud(tx)).to be false
20      end
21    end
22  end
23 end
```

Dans cet exemple, le composant IA FraudDetector est testé avec deux cas de test : un pour une transaction frauduleuse et un autre pour une transaction légitime. Les cas de test vérifient que la méthode `detect_fraud` renvoie la valeur booléenne attendue en fonction des caractéristiques de la transaction.

Test de Bout en Bout

Le test de bout en bout implique de tester l'ensemble du flux de travail du début à la fin, en simulant des scénarios réels et des interactions utilisateur. Il garantit que le flux de travail se comporte correctement et produit les résultats souhaités. Lors de l'exécution de tests de bout en bout pour les flux de travail intelligents, considérez les points suivants :

1. **Scénarios Utilisateur** : Identifiez les scénarios utilisateur courants et testez le comportement du flux de travail dans ces scénarios. Vérifiez que le flux de travail

traite correctement les entrées utilisateur, prend les décisions appropriées et produit les résultats attendus.

2. Validation des Données : Assurez-vous que le flux de travail valide et nettoie les entrées utilisateur pour éviter les incohérences de données ou les vulnérabilités de sécurité. Testez le flux de travail avec différents types de données d'entrée, y compris des données valides et invalides.

3. Récupération d'Erreurs : Testez la capacité du flux de travail à récupérer des erreurs et des exceptions. Simulez des scénarios d'erreur et vérifiez que le flux de travail les gère élégamment, enregistre les erreurs et prend les mesures de récupération appropriées.

4. Performance et Évolutivité : Évaluez la performance et l'évolutivité du flux de travail dans différentes conditions de charge. Testez le flux de travail avec un grand volume de requêtes simultanées et mesurez les temps de réponse, l'utilisation des ressources et la stabilité globale du système.

Voici un exemple de test de bout en bout pour un flux de travail en Ruby utilisant le framework de test RSpec et la bibliothèque Capybara pour simuler les interactions utilisateur :

```
1 RSpec.describe 'Order Processing Workflow' do
2   scenario 'User places an order successfully' do
3     visit '/orders/new'
4     fill_in 'Product', with: 'Sample Product'
5     fill_in 'Quantity', with: '2'
6     fill_in 'Shipping Address', with: '123 Main St'
7     click_button 'Place Order'
8
9     expect(page).to have_content('Order Placed Successfully')
10    expect(Order.count).to eq(1)
11    expect(Order.last.status).to eq('processed')
12  end
13end
```

Dans cet exemple, le test de bout en bout simule un utilisateur passant une commande via l'interface web. Il remplit les champs de formulaire requis, soumet la commande et

vérifie que la commande est traitée avec succès, affichant le message de confirmation approprié et mettant à jour le statut de la commande dans la base de données.

Intégration et Déploiement Continus

Pour garantir la fiabilité et la maintenabilité des flux de travail intelligents, il est recommandé d'intégrer les tests et la validation dans le pipeline d'intégration et de déploiement continu (CI/CD). Cela permet l'automatisation des tests et la validation des modifications du flux de travail avant leur déploiement en production. Considérez les pratiques suivantes :

- 1. Exécution Automatisée des Tests** : Configurez le pipeline CI/CD pour exécuter automatiquement la suite de tests chaque fois que des modifications sont apportées au code du flux de travail. Cela garantit que toute régression ou défaillance est détectée tôt dans le processus de développement.
- 2. Surveillance de la Couverture des Tests** : Mesurez et surveillez la couverture des tests des composants du flux de travail et des points de décision IA. Visez une couverture de tests élevée pour garantir que les chemins et scénarios critiques sont minutieusement testés.
- 3. Retour Continu** : Intégrez les résultats des tests et les métriques de qualité du code dans le flux de développement. Fournissez un retour continu aux développeurs sur l'état des tests, la qualité du code et tout problème détecté pendant le processus CI/CD.
- 4. Environnements de Préproduction** : Déployez le flux de travail dans des environnements de préproduction qui reflètent fidèlement l'environnement de production. Effectuez des tests et des validations supplémentaires dans l'environnement de préproduction pour détecter tout problème lié à l'infrastructure, à la configuration ou à l'intégration des données.
- 5. Mécanismes de Restauration** : Mettez en place des mécanismes de restauration en cas d'échec du déploiement ou de problèmes critiques détectés en production. Assurez-

vous que le flux de travail peut être rapidement restauré à une version stable précédente pour minimiser les temps d'arrêt et l'impact sur les utilisateurs.

En incorporant les tests et la validation tout au long du cycle de développement des flux de travail intelligents, les organisations peuvent garantir la fiabilité, l'exactitude et la maintenabilité de leurs systèmes alimentés par l'IA. Des tests et des validations réguliers aident à détecter les bugs, prévenir les régressions et renforcer la confiance dans le comportement et les résultats du flux de travail.

Partie 2 : Les Patrons

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Ingénierie des Prompts

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Chaîne de réflexion

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça marche

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemples

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Génération de contenu

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Création d'Entités Structurées

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Guide des agents LLM

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Avantages et considérations

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Mode Switch

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Quand l'utiliser

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Attribution de Rôle

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment Cela Fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Quand l'Utiliser

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemples

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Prompt Object

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Modèle de Prompt

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Avantages et considérations

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Quand l'utiliser :

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Entrées/Sorties structurées

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Mise à l'échelle des entrées/sorties structurées

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Avantages et considérations

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Chaînage de prompts

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Quand l'utiliser

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple : L'intégration d'Olympia

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Réécriture de Prompts

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment Ça Fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Délimitation des réponses

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Avantages et Considérations

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Gestion des erreurs

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Query Analyzer

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Mise en œuvre

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Étiquetage morpho-syntaxique (POS) et Reconnaissance d'entités nommées (NER)

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Classification d'intention

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Extraction de mots-clés

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Avantages

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Réécriveur de requêtes

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Avantages

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Ventriloquist

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Quand l'utiliser

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Composants Discrets

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Prédicat

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment Ça Fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Quand l'utiliser

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Facade d'API

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Principaux avantages

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Quand l'utiliser

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Authentification et Autorisation

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Gestion des Requêtes

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Formatage des Réponses

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Gestion des Erreurs et Cas Limites

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Considérations d'Extensibilité et de Performance

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comparaison avec d'Autres Patrons de Conception

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Result Interpreter

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Quand l'utiliser

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Virtual Machine

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Quand l'utiliser

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Derrière la magie

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Spécification et Tests

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Spécifier le Comportement

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Écriture des Cas de Test

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple : Test du Composant Traducteur

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Selecture des interactions HTTP

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Human In The Loop (HITL)

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Modèles de Haut Niveau

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Intelligence Hybride

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Réponse Adaptative

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Alternance des rôles Humain-IA

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Escalade

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Avantages clés

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Application concrète : Santé

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Boucle de rétroaction

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça marche

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Applications et Exemples

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Techniques avancées d'intégration des retours humains

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Radiation Passive d'Information

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment Cela Fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Affichage Contextuel d'Information

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Notifications Proactives

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Aperçus Explicatifs

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exploration Interactive

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Avantages Clés

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Applications et Exemples

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Prise de Décision Collaborative (PDC)

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment Cela Fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Apprentissage Continu

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Applications et Exemples

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Considérations éthiques

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Rôle du HITL dans l'atténuation des risques liés à l'IA

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Avancées technologiques et perspectives d'avenir

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Défis et limites des systèmes HITL

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Gestion intelligente des erreurs

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Approches traditionnelles de gestion des erreurs

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Diagnostic contextuel des erreurs

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça marche

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Ingénierie des invites pour le diagnostic contextuel des erreurs

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Génération augmentée par récupération pour le diagnostic d'erreur contextuel

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Rapport d'erreur intelligent

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Prévention prédictive des erreurs

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Récupération Intelligente des Erreurs

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Communication Personnalisée des Erreurs

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment Cela Fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Flux de Travail Adaptatif de Gestion des Erreurs

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Contrôle Qualité

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Eval

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Problème

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Solution

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Fonctionnement

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Considérations

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comprendre les références étalons

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment fonctionnent les évaluations sans référence

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Garde-fou

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Problème

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Solution

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Comment ça fonctionne

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Exemple

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Considérations

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Garde-fous et Évaluations : Les Deux Faces d'une Même Médaille

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

L'Interchangeabilité des Garde-fous et des Évaluations Sans Référence

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Mise en œuvre de Garde-fous et d'Évaluations à Double Usage

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Glossaire

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Glossaire

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

A

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

B

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

C

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

D

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

E

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

F

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

G

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

H

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

I

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

J

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

K

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

L

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

M

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

N

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

O

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

P

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Q

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

R

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

S

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

T

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

U

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

V

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

W

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Z

Ce contenu n'est pas disponible dans le livre d'échantillons. Le livre peut être acheté sur Leanpub à <http://leanpub.com/patterns-of-application-development-using-ai-fr>.

Index

- accessibilité, 222, 223
- Agentique, 33
- agents conversationnels de service client,
 - 34
- AI, 101, 146, 207
 - model, 100, 101
- Aide à la décision clinique, 105
- ajustement fin, 82
- ajustement par instructions, 10
 - modèles ajustés par instructions, 51
- algèbre linéaire, 44
- Alpaca, 13
- Altman, Sam, 18
- Amazon Web Services, 258
- amorçage à peu d'exemples, 64
- analyse des sentiments, 17, 102, 114, 115,
 - 117, 120, 121, 138, 148
- Anthropic, 23, 40, 75, 132, 140
- anthropomorphisme, 70
- APIs, 73, 126, 156
- appel d'outil, 156
- appel de fonction
 - échec, 137
- application d'agent conversationnel, 121
- Applications E-commerce, 93
- applications modernes, 229
- applications éducatives, 33
- apprentissage non supervisé, 4
- apprentissage sans exemple, 61
- apprentissage zero-shot, 60
- apprentissage à peu d'exemples, 63
- Apprentissage à un exemple, 62
- architecture d'applications d'entreprise, 39
- architecture distribuée, 255
- Architecture en microservices, 91
- architecture logicielle, 2
- architecture transformer, 6
- assistants virtuels, 34
- attribution de ticket, 245
- audit et conformité, 253
- audit logging, 109
- Auto-continuation, 163
- balisage de type markup, 72
- base de connaissances d'Olympia, 93
- bases de connaissances, 8
- bases de données, 126
 - objet supporté, 107
- stratégies de verrouillage, 112
- BERT, 14, 24
- biais
 - et équité dans l'IA, 263
- bibliothèque Capybara, 265

- boundary conditions, 260
Brotli, 259
Byte Pair Encoding (BPE), 15
- C (Langage de programmation), 119
cadres de développement, 151
cas limites, 59
Chain of Thought (CoT), 142
ChatGPT, 31, 54
chaînage de workers IA, 114
chaîne d'approvisionnement
 optimisation, 33
Chaîne de Pensée (CoT), 46
classes, 36
classification, 54, 123
Claude, 8, 45, 79
Claude 3, 51, 130, 132, 138, 140
Claude 3 Opus, 76
Claude v1, 17
Claude v2, 17
Cohere (fournisseur de LLM), 23
Cohere (Fournisseur LLM), 25
cohérence
 et reproductibilité, 136
Collecte des antécédents médicaux, 103
commerce électronique, 227
complétion haute performance, 27
comportement déterministe, 59
compte, 93
conception d'applications et frameworks,
 203
concurrent workflows, 259
- confiance des utilisateurs, 223
connectivité réseau, 232
conservation et rotation des journaux, 254
construction narrative, 20
contenu
 Catégorisation du contenu, 114
 filtrage, 27
contenu généré par l'utilisateur, 114
context
 Contextual Content Generation, 196,
 197
 entrées infiniment longues, 16
 window, 15
contexte
 Augmentation, 47
 fenêtre, 231
 Génération de Contenu Contextuel,
 191, 204, 205
 prise de décision contextuelle, 231
 Suggestions contextuelles de champs,
 206
conversation
 boucle, 160, 162
 transcription, 160, 162
correspondance de motifs, 155
Customer Sentiment Analysis, 101
- data
 persistence, 112
 preparation, 111
Datadog, 254
decision

- making capabilities, 101
- dictionnaires, 134
- directive système, 131
- divulgation progressive, 212
- document clustering, 123
- Dohan, et al., 45
- données
 - analyse, 35, 150
 - confidentialité, 27, 221
 - flux, 113
 - intégrité, 246
 - pipeline de traitement, 246
 - Récupération des données, 112
 - Synchronisation des données, 112
 - tâches de traitement, 129
 - Validation des Données, 265
- Données auto-réparatrices, 167, 249
- données d'entraînement, 43
- données en flux continu, 155
- données structurées, 137
- débit, 28
- débogage, 231
 - et dépannage, 252
 - et tests, 135
- décision
 - arbres, 227
 - cas d'utilisation, 136
 - points, 250
- découvertes médicales, 102
- défis conceptuels et pratiques, 204
- Délimitation des réponses, 210
- détaillants en ligne, 210
- détection de fraude
 - système, 99
- développement d'applications, 226
- e-commerce, 196
- E/S structurées, 210
- efficacité, 228
- employés de Databricks, 54
- Encodage par paires d'octets (BPE), 13
- ensembles, 120, 121
 - ensemble d'agents, 121
- entrée
 - prompts, 57
- environnements de développement locaux, 158
- environnements de préproduction, 266
- erreurs
 - gestion, 112, 145
 - récupération, 265
 - taux, 113
- erreurs de syntaxe, 134
- errors
 - handling, 109, 260
 - Intelligent Error Handling, 146
- espace latent, 41, 43
- event-driven architecture, 111
- explicabilité, 263
- expérience utilisateur, 199
- expérimentation
 - cadre, 198
- extensibilité, 228, 254
- exécution parallèle, 255

- F#, 95
- Facebook, 25
- facteurs de risque, 97, 98
- filtrage basé sur le contenu, 94
- filtrage collaboratif, 94
- FitAI, 216
- flexibilité et créativité, 201
- flux de travail adaptatif
- Composition Adaptative des Flux de Travail, 231
- flux de travail multi-étapes, 114
- fonction
- appel, 126, 127, 160
 - historique des appels, 159
 - noms, 157
- fournisseurs d'hébergement de modèles
- open source, 210
- Gemma 7B, 11
- Generative Pre-trained Transformer (GPT),
- 8
- gestion des connaissances, 33
- gestion des exceptions, 232, 234
- gestion du trafic, 33
- Gestionnaire de Processus, 106
- gestionnaires de flux, 154
- GitLab, 95
- Global Interpreter Lock (GIL), 118
- Google, 23
- API, 64, 66
 - Cloud Platform, 258
 - Gemini, 22
- Gemini 1.5 Pro, 14, 17, 19
- PaLM (Modèle de Langage Pathways),
- 24
- PaLM (Pathways Language Model), 17
- Plateforme Cloud AI, 24
- T5, 14
- goulets d'étranglement, 231
- GPT-3, 13, 17
- GPT-4, 6, 13, 17, 22, 32, 45, 51, 64, 107, 120, 122, 130, 136, 208, 209, 256
- Graham, Paul, 19
- Grand Modèle de Langage (GML), 1, 3, 18, 30, 78, 113, 137, 147, 191, 203
- Grand modèle de langage (GML), 73, 89, 128, 143, 150, 208, 237
- Grand Modèle de Langage (LLM), 126
- paysage, 28
- Grand modèle de langage (LLM), 68, 70, 79, 167, 170, 214
- GraphQL, 110
- Groq, 26, 122
- gzip, 259
- Génération Augmentée par Recherche (RAG), 32
- Génération Augmentée par Récupération (RAG), 39, 47
- Génération augmentée par récupération (RAG), 81, 128
- génération de données synthétiques, 54
- génération dynamique d'IU, 192
- génération intermodale, 22

- hardware, 29
- Hohpe, Gregor, 106
- Honeybadger, 96
- HTTP, 153
- Human-In-The-Loop (HITL) (Humain dans la boucle), 183
- hyperparamètre, 48
- IA, 66, 76, 131, 137, 153, 215
 - applications, 128, 141, 152, 165
 - conversationnelle, 7, 32, 217
 - modèle, 91, 158, 159, 161, 216
 - points de décision, 263
 - systèmes composés, 31, 35
- information
 - extraction, 54
 - récupération, 7, 129
- informatique, 72, 74
- Inférence, 5
- injections SQL, 72
- input
 - validation, 260
- instruction tuning
 - modèles ajustés par instructions, 53
- intelligent workflow orchestration, 259
- interactions de type jeu de rôle, 7
- Interface Utilisateur (IU)
 - conception, 224
- Interface utilisateur (IU)
 - interfaces, 203
 - technologies, 214
- Interface utilisateur (UI)
 - frameworks, 220
 - interfaces, 219
- interface utilisateur adaptative, 213
- Interface Utilisateur Générative (GenUI), 223
- Interface utilisateur générative (GenUI), 203, 210, 211, 215, 219
- interface visuelle, 214
- interfaces contrôlées par la voix, 34
- interfaces inclusives, 204
- internationalisation, 199
- intervention manuelle, 234
- intégration des GML, 192
- Intégration et Déploiement Continus (CI/CD), 266
 - pipeline, 266
- invites
 - ingénierie, 42
- iterative refinement, 147
- journalisation granulaire, 253
- journalisation structurée, 253
- JSON (JavaScript Object Notation), 130, 134, 138, 150, 170
- K-means, 124
- langage
 - modèles, 67, 75
 - tâches linguistiques, 5
- langage naturel
 - Traitemet du Langage Naturel (TLN), 103

- language 7B, 11
models, 44
- langue
Détection de la langue, 114
- Large Language Model (LLM), 16, 123, 147
- latence, 28
- Latent Dirichlet Allocation, 124
- ligne de commande
Interface en Ligne de Commande
(CLI), 26
- Llama, 13
- Llama 2-70B, 51
- Llama 3 70B, 11
- Llama 3 8B, 11
- logique de disjoncteur, 165
- Louvre, 44
- lunettes de réalité augmentée, 224
- Managed Streaming for Apache Kafka, 42
- Markdown, 150
- Memorial Sloan Kettering Cancer Center, 42
- Mercure (dieu romain), 46
- Mercure (planète), 46
- Mercure (élément), 46
- message déclencheur, 106
- MessagePack, 259
- Meta, 25
- Metropolitan Museum of Art, 44
- mise en cache, 256
- mise à l'échelle automatique, 258
- Mistral, 26
- Mixtral
8x22B, 11
8x7B, 57
- modularité, 90
- modèle OPT, 25
- modèles basés sur la récupération, 7
- modèles clés, 229
- Modèles d'intégration d'entreprise, 106
- modèles de base, 55
- modèles graphiques, 45
- modèles historiques, 231
- modèles probabilistes, 44
- modélisation autorégressive, 44
- Modérateur de Contenu Intelligent, 239
- Multi-Agent
Résolveurs de Problèmes, 32
- Multimodal
modèles, 20
modèles de langage, 21
- Multitude d'agents, 169
- Multitude de Travailleurs, 121
- mécanismes de nouvelle tentative, 112
- mécanismes de restauration, 266
- méthode finalize, 159, 161, 162
- Naive Bayes, 124
- natural language
Natural Language Processing (NLP), 123
- Nettoyage du texte, 114

- New Relic, 257
- Ollama, 26
- Olympia, 34, 64, 131, 146, 154, 170
- OpenAI, 3, 23, 40, 75
- OpenRouter, 28, 29, 154, 256
- orchestration des flux de travail
- intelligents, 234
- orchestration intelligente des flux de travail, 226, 256
- ordinateurs de bureau, 224
- output verification, 260
- paramètre
- effets, 132
 - gamme, 11
 - Nombre de Paramètres, 29
- paramètres
- d'entrée, 131
- paraphrasage, 54
- paysage numérique, 198
- performance
- compromis, 5
 - optimisation, 136, 201, 252
 - problèmes, 257
- Perplexity (Fournisseur), 12
- personnalisation, 27, 193, 223, 228
- Formulaires Personnalisés, 205
 - Micro-texte personnalisé, 211
- pile ELK, 113
- planification des interventions d'urgence, 33
- principe du moindre privilège, 73
- problèmes d'utilisabilité, 222
- Process Manager, 109
- Enterprise Integration, 235
- processus de distillation, 78
- Productivité, 194
- programmation fonctionnelle, 94
- prompts
- chaînage, 60, 73
 - conception, 59, 69
 - Distillation de Prompts, 47, 75
 - Distillation de prompts, 256
 - Distillation des prompts, 80
 - ingénierie, 46, 47, 57, 61, 66, 68, 220
 - Modèle de prompt, 60, 210
 - Prompt Object, 76
 - raffinement, 70
- propriétés ACID, 112
- Protocol Buffers, 259
- prédictions, 6
- psychologie de l'utilisateur, 221
- publish-subscribe systems, 110
- PyTorch, 25
- Pénalité de présence, 50
- pénalités de répétition, 53
- Quantification, 29
- questions-réponses fermées et ouvertes, 54
- Qwen2 70B, 11
- raffinement itératif, 78
- Rails, 199
- Railway Oriented Programming (ROP), 97
- Raix, 235

- bibliothèque, 99
- Recommandations de Produits, 94
- recommandations de produits
 - personnalisées, 94
- Response Fencing (clôture de réponse), 180
- restreindre le chemin, 39, 40
- Result Interpreter, 145
- Routage dynamique des tâches, 229
- RSpec, 260, 262, 265
- Ruby, 95, 96, 115, 166, 265
- Ruby on Rails, 1, 114, 235, 243
- Rudall, Alex, 24
- Rust (Langage de programmation), 119
- Rust (Programming Language), 95
 - règles grammaticales, 4
 - règles métier, 227
 - régression linéaire, 44
 - réseaux neuronaux, 4, 6
 - rétroaction
 - Boucle de rétroaction, 60
 - sans état, 160
- Scout, 257
- services externes ou API, 129
- smartphones, 224
- Stratification des risques, 105
- stratégies de motivation, 219
- stratégies de repli, 112
- stratégies de segmentation et de ciblage,
 - 198
- Stripe, 132
- suivi des métriques clés, 250
- support client, 33
- Support Vector Machines (SVM), 124
- surveillance
 - et alertes, 233
 - et journalisation, 113, 252
 - métriques, 253
- Surveillance continue des risques, 105
- synthèse, 54
- system directive, 100
- systèmes de questions-réponses, 8
- Sélection Dynamique d’Outils, 134
- Sélection Forcée d’Outils, 135
- T5, 24
- table de hachage, 155
- tableaux, 134
- tablettes, 224
- Temperature, 55
- temps de traitement, 113
- Temps jusqu’au Premier Jeton (TTFT), 28
- test de bout en bout, 264, 265
- tests d’intégration, 261
- tests utilisateur et retours, 201
- théorie de l’esprit, 41
- Together.ai, 26
- tokenisation, 13
- tokens, 6, 13
- ton émotionnel, 148
- topic identification, 123
- traduction, 17, 200
- tragédie des biens communs, 195
- traitement asynchrone, 255

- traitement de flux, 153, 159, 165
logique, 161
- traitement par lots, 256
- Transformeur génératif pré-entraîné (GPT),
68
- tâches complexes, 149
- Unicode-encodable language, 15
- Universal ID, 259
- utilisation d'outils, 126, 127, 152
- Ventriloquist (ventriloque), 180
- verrouillage optimiste, 112
- verrouillage pessimiste, 112
- vote majoritaire, 120
- Vérification d'assurance, 103
- Wall, Larry, 3
- Wisper, 96, 109, 154, 161
- Wooley, Chad, 95
- XML, 137
- Yi-34B, 51
- Échantillonnage top-k, 49
- Échantillonnage top-p (nucleus), 49
- Évaluation et stratification des symptômes,
103
- écosystème, 151
- écriture créative, 35, 54
- éthique
implications, 204
- événements envoyés par le serveur (SSE),
153