

Patterns of Application Development Using AI

Obie Fernandez
Foreword by Gregor Hohpe



Leanpub

Edisi Bahasa Indonesia

Pola Pengembangan Aplikasi Menggunakan AI (Edisi Bahasa Indonesia)

Obie Fernandez

Buku ini dijual di <http://leanpub.com/patterns-of-application-development-using-ai-id>

Versi ini diterbitkan pada 2025-01-23



Leanpub

Ini adalah sebuah buku [Leanpub](#). Leanpub memberdayakan penulis dan penerbit dengan proses Lean Publishing. [Lean Publishing](#) adalah model penerbitan ebook dalam-proses menggunakan piranti ringan dan sejumlah iterasi untuk memperoleh masukan dari pembaca, menerapkan pivot hingga Anda dapat mewujudkan komposisi buku yang pas dan menarik.

© 2025 Obie Fernandez

Tweet Buku Ini!

Bantulah Obie Fernandez dengan mewartakan buku ini via [Twitter!](#)

Tagar yang disarankan untuk buku ini adalah [#poaduai](#).

Temukan kata orang tentang buku ini dengan mengklik tautan ini untuk menampilkan tagar ini di Twitter:

[#poaduai](#)

Untuk ratuku yang tangguh, musaku, cahaya dan cintaku, Victoria

Juga Oleh **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

Contents

Kata Pengantar oleh Gregor Hohpe	i
Kata Pengantar	ii
Tentang Buku Ini	iii
Tentang Contoh Kode	iii
Yang Tidak Saya Bahas	iii
Untuk Siapa Buku Ini	iii
Membangun Kosakata Umum	iii
Cara Terlibat	iii
Ucapan Terima Kasih	iii
Bagaimana dengan ilustrasinya?	iv
Tentang Lean Publishing	iv
Tentang Penulis	v
Pendahuluan	1
Pemikiran tentang Arsitektur Perangkat Lunak	2
Apa itu Model Bahasa Besar?	3
Memahami Inferensi	5
Mempertimbangkan Kinerja	27
Bereksperimen Dengan Model LLM yang Berbeda	29
Sistem AI Majemuk	30

Bagian 1: Pendekatan & Teknik Fundamental	38
Mempersempit Jalur	39
Ruang Laten: Sangat Luas Tak Terpenuhi	41
Bagaimana Jalur Tersebut “Dipersempit”	45
Model Mentah Versus Model yang Dilatih dengan Instruksi	49
Rekayasa Prompt	57
Distilasi Prompt	74
Bagaimana dengan fine-tuning?	81
Retrieval Augmented Generation (RAG)	83
Apa itu Retrieval Augmented Generation?	83
Bagaimana RAG Bekerja?	83
Mengapa Menggunakan RAG dalam Aplikasi Anda?	83
Mengimplementasikan RAG dalam Aplikasi Anda	83
Pemecahan Proposisi	84
Contoh RAG dalam Dunia Nyata	84
Intelligent Query Optimization (IQO)	85
Perangkingan Ulang	85
Penilaian RAG (RAGAs)	85
Tantangan dan Prospek Masa Depan	87
Banyaknya Pekerja	89
Pekerja AI Sebagai Komponen Mandiri yang Dapat Digunakan Kembali	90
Pengelolaan Akun	92
Aplikasi E-commerce	93
Aplikasi Kesehatan	102
AI Worker sebagai Process Manager	106
Mengintegrasikan AI Worker ke Dalam Arsitektur Aplikasi Anda	109

CONTENTS

Kemampuan Komposisi dan Orkestrasi Pekerja AI	113
Menggabungkan NLP Tradisional dengan LLM	122
Penggunaan Alat	125
Apa itu Penggunaan Alat?	125
Potensi Penggunaan Alat	127
Alur Kerja Penggunaan Alat	128
Praktik Terbaik untuk Penggunaan Tool	143
Menyusun dan Merantai Alat	147
Arah Masa Depan	149
Pemrosesan Stream	152
Mengimplementasikan ReplyStream	153
“Putaran Percakapan”	159
Kelanjutan Otomatis	162
Kesimpulan	164
Data yang Dapat Memperbaiki Diri	166
Studi Kasus Praktis: Memperbaiki JSON yang Rusak	169
Pertimbangan dan Kontraindikasi	174
Pembuatan Konten Kontekstual	189
Personalisasi	190
Produktivitas	192
Iterasi dan Eksperimen Cepat	194
Lokalisasi Bertenaga AI	197
Pentingnya Pengujian Pengguna dan Umpan Balik	199
Generative UI	200
Menghasilkan Teks untuk Antarmuka Pengguna	202
Mendefinisikan UI Generatif	211

CONTENTS

Contoh	213
Peralihan ke Desain Berorientasi Hasil	216
Tantangan dan Pertimbangan	217
Prospek dan Peluang Masa Depan	219
Orkestrasi Alur Kerja Cerdas	223
Kebutuhan Bisnis	224
Manfaat Utama	225
Pola-pola Utama	225
Penanganan dan Pemulihan Pengecualian	228
Mengimplementasikan Orkestrasi Alur Kerja Cerdas dalam Praktik	231
Pemantauan dan Pencatatan	246
Pertimbangan Skalabilitas dan Kinerja	250
Pengujian dan Validasi Alur Kerja	255
Bagian 2: Pola-Pola	264
Rekayasa Prompt	265
Rantai Pemikiran	266
Pengalihan Mode	267
Penugasan Peran	268
Objek Prompt	269
Prompt Template	270
IO Terstruktur	271
Perangkaian Prompt	272
Penulis Ulang Prompt	273
Response Fencing	274
Query Analyzer	275
Pengolah Ulang Kueri	276
Ventriloquist	277

CONTENTS

Komponen Diskret	278
Predicate	279
API Facade	280
Penerjemah Hasil	282
Mesin Virtual	283
Spesifikasi dan Pengujian	283
Human In The Loop (HITL)	285
Pola-Pola Tingkat Tinggi	285
Eskalasi	286
Siklus Umpan Balik	287
Radiasi Informasi Pasif	288
Collaborative Decision Making (CDM)	290
Pembelajaran Berkelanjutan	291
Pertimbangan Etis	291
Kemajuan Teknologi dan Prospek Masa Depan	291
Penanganan Kesalahan Cerdas	293
Pendekatan Tradisional dalam Penanganan Kesalahan	293
Diagnosis Kesalahan Kontekstual	294
Pelaporan Kesalahan Cerdas	295
Pencegahan Error Prediktif	296
Pemulihan Kesalahan Cerdas	296
Komunikasi Kesalahan Terpersonalisasi	297
Alur Kerja Penanganan Kesalahan Adaptif	298
Kendali Mutu	299
Eval	300
Pagar Pengaman	302
Guardrails dan Evaluasi: Dua Sisi Mata Uang yang Sama	302

Glosarium	304
Glosarium	304
Index	309

Kata Pengantar oleh Gregor Hohpe

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kata Pengantar

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Tentang Buku Ini

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Tentang Contoh Kode

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Yang Tidak Saya Bahas

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Untuk Siapa Buku Ini

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Membangun Kosakata Umum

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Terlibat

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Ucapan Terima Kasih

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Bagaimana dengan ilustrasinya?

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

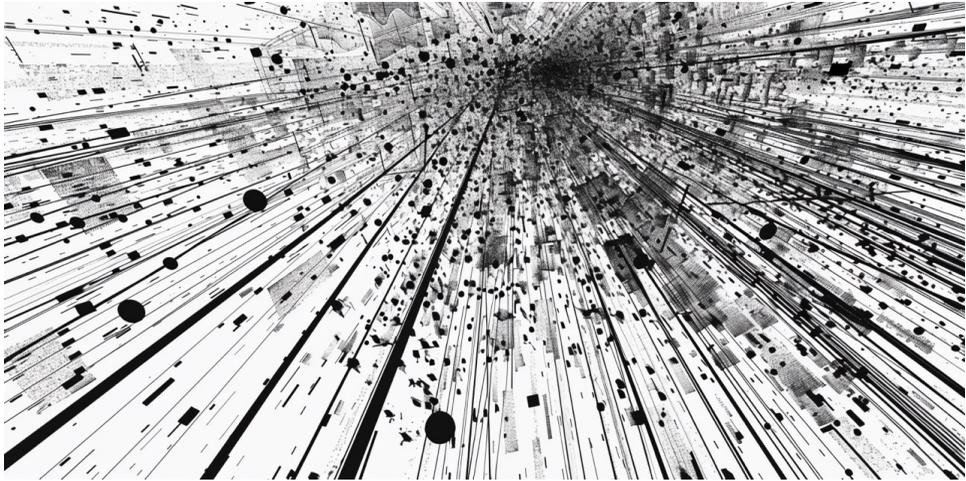
Tentang Lean Publishing

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Tentang Penulis

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pendahuluan



Jika Anda tidak sabar untuk mulai mengintegrasikan Model Bahasa Besar (LLM) AI ke dalam proyek pemrograman Anda, silakan langsung mempelajari pola dan contoh kode yang disajikan di bab-bab selanjutnya. Namun, untuk benar-benar memahami kekuatan dan potensi pola-pola ini, ada baiknya meluangkan waktu sejenak untuk memahami konteks yang lebih luas dan pendekatan terpadu yang mereka representasikan.

Pola-pola ini bukan sekadar kumpulan teknik yang terisolasi, melainkan sebuah kerangka kerja terpadu untuk mengintegrasikan AI ke dalam aplikasi Anda. Saya menggunakan Ruby on Rails, tetapi pola-pola ini seharusnya dapat bekerja di hampir semua lingkungan pemrograman lainnya. Mereka menangani berbagai masalah, mulai dari manajemen data dan optimasi kinerja hingga pengalaman pengguna dan keamanan, menyediakan perangkat lengkap untuk meningkatkan praktik pemrograman tradisional dengan kemampuan AI.

Setiap kategori pola mengatasi tantangan atau peluang spesifik yang muncul ketika memasukkan komponen AI ke dalam aplikasi Anda. Dengan memahami hubungan dan sinergi antara pola-pola ini, Anda dapat membuat keputusan yang tepat tentang di mana dan bagaimana menerapkan AI secara paling efektif.

Pola-pola ini tidak pernah menjadi solusi preskriptif dan seharusnya tidak diperlakukan demikian. Mereka dimaksudkan sebagai blok pembangun yang dapat disesuaikan dengan kebutuhan dan batasan unik aplikasi Anda sendiri. Penerapan pola-pola ini yang berhasil (seperti pola lainnya dalam bidang perangkat lunak) bergantung pada pemahaman mendalam tentang ranah masalah, kebutuhan pengguna, dan arsitektur teknis keseluruhan proyek Anda.

Pemikiran tentang Arsitektur Perangkat Lunak

Saya mulai memprogram pada tahun 1980-an dan terlibat dalam komunitas peretas, dan tidak pernah kehilangan pola pikir peretas saya, bahkan setelah menjadi pengembang perangkat lunak profesional. Sejak awal, saya selalu memiliki skeptisisme yang sehat tentang nilai apa yang sebenarnya diberikan oleh para arsitek perangkat lunak di menara gading mereka.

Salah satu alasan mengapa saya secara pribadi sangat bersemangat tentang perubahan yang dibawa oleh gelombang baru teknologi AI yang kuat ini adalah dampaknya terhadap apa yang kita anggap sebagai keputusan *arsitektur perangkat lunak*. Ini menantang gagasan tradisional tentang apa yang dianggap cara yang “benar” untuk merancang dan mengimplementasikan proyek perangkat lunak kita. Ini juga mempertanyakan apakah arsitektur masih dapat dianggap terutama sebagai *bagian-bagian sistem yang sulit diubah*, mengingat peningkatan AI membuatnya lebih mudah dari sebelumnya untuk mengubah bagian apa pun dari proyek Anda, kapan saja.

Mungkin kita sedang memasuki tahun-tahun puncak pendekatan “post-modern” dalam rekayasa perangkat lunak. Dalam konteks ini, post-modern mengacu pada pergeseran

fundamental dari paradigma tradisional, di mana pengembang bertanggung jawab untuk menulis dan memelihara setiap baris kode. Sebaliknya, pendekatan ini menganut gagasan mendelegasikan tugas-tugas, seperti manipulasi data, algoritma kompleks, dan bahkan bagian-bagian lengkap logika aplikasi, kepada pustaka pihak ketiga dan API eksternal. Pergeseran post-modern ini merepresentasikan perubahan signifikan dari kebijaksanaan konvensional dalam membangun aplikasi dari nol, dan menantang pengembang untuk memikirkan kembali peran mereka dalam proses pengembangan.

Saya selalu percaya bahwa programmer yang baik hanya menulis kode yang benar-benar perlu ditulis, berdasarkan ajaran Larry Wall dan tokoh-tokoh hacker terkemuka lainnya seperti dia. Dengan meminimalkan jumlah kode yang ditulis, kita bisa bergerak lebih cepat, mengurangi area kemungkinan bug, menyederhanakan pemeliharaan, dan meningkatkan keandalan aplikasi secara keseluruhan. Kode yang lebih sedikit memungkinkan kita untuk fokus pada logika bisnis inti dan pengalaman pengguna, sambil mendelegasikan pekerjaan lain ke layanan lain.

Sekarang sistem berbasis AI dapat menangani tugas-tugas yang sebelumnya hanya bisa dilakukan oleh kode yang ditulis manusia, kita seharusnya bisa menjadi lebih produktif dan tangkas, dengan fokus yang lebih besar dari sebelumnya pada penciptaan nilai bisnis dan pengalaman pengguna.

Tentu saja ada pertukaran dalam mendelegasikan bagian besar proyek Anda ke sistem AI, seperti potensi kehilangan kontrol, dan kebutuhan akan mekanisme pemantauan dan umpan balik yang kuat. Itulah mengapa hal ini membutuhkan serangkaian keterampilan dan pengetahuan baru, termasuk setidaknya beberapa pemahaman dasar tentang cara kerja AI.

Apa itu Model Bahasa Besar?

Model Bahasa Besar (LLM) adalah jenis model kecerdasan buatan yang telah mendapatkan perhatian signifikan dalam beberapa tahun terakhir, sejak peluncuran GPT-3

oleh OpenAI pada tahun 2020. LLM dirancang untuk memproses, memahami, dan menghasilkan bahasa manusia dengan akurasi dan kelancaran yang luar biasa. Dalam bagian ini, kita akan melihat secara singkat bagaimana LLM bekerja dan mengapa mereka sangat cocok untuk membangun komponen sistem cerdas.

Pada intinya, LLM didasarkan pada algoritma pembelajaran mendalam, khususnya jaringan saraf. Jaringan ini terdiri dari node-node yang saling terhubung, atau neuron, yang memproses dan mengirimkan informasi. Arsitektur pilihan untuk LLM seringkali adalah model Transformer, yang terbukti sangat efektif dalam menangani data sekuensial seperti teks.

Model Transformer didasarkan pada mekanisme attention dan terutama digunakan untuk tugas-tugas yang melibatkan data sekuensial, seperti pemrosesan bahasa alami. Transformer memproses data masukan sekaligus alih-alih secara sekuensial, yang memungkinkan mereka menangkap dependensi jarak jauh dengan lebih efektif. Mereka memiliki lapisan-lapisan mekanisme attention yang membantu model fokus pada berbagai bagian data masukan untuk memahami konteks dan hubungan.

Proses pelatihan LLM melibatkan pemaparan model terhadap sejumlah besar data tekstual, seperti buku, artikel, situs web, dan repositori kode. Selama pelatihan, model belajar mengenali pola, hubungan, dan struktur dalam teks. Model menangkap properti statistik bahasa, seperti aturan tata bahasa, asosiasi kata, dan makna kontekstual.

Salah satu teknik utama yang digunakan dalam pelatihan LLM adalah pembelajaran tanpa pengawasan. Ini berarti model belajar dari data tanpa pelabelan atau panduan eksplisit. Model menemukan pola dan representasi secara mandiri dengan menganalisis ko-kemunculan kata dan frasa dalam data pelatihan. Hal ini memungkinkan LLM mengembangkan pemahaman mendalam tentang bahasa dan kerumitannya.

Aspek penting lainnya dari LLM adalah kemampuan mereka dalam menangani *konteks*. Ketika memproses sebuah teks, LLM tidak hanya mempertimbangkan kata-kata individual tetapi juga konteks di sekitarnya. Mereka memperhitungkan kata-kata, kalimat, dan bahkan paragraf sebelumnya untuk memahami makna dan maksud teks.

Pemahaman kontekstual ini memungkinkan LLM menghasilkan respons yang koheren dan relevan. Salah satu cara utama kita mengevaluasi kemampuan model LLM tertentu adalah dengan mempertimbangkan ukuran konteks yang dapat mereka pertimbangkan untuk menghasilkan respons.

Setelah dilatih, LLM dapat digunakan untuk berbagai tugas terkait bahasa. Mereka dapat menghasilkan teks seperti manusia, menjawab pertanyaan, meringkas dokumen, menerjemahkan bahasa, dan bahkan menulis kode. Kesenjangan LLM membuat mereka berharga untuk membangun komponen sistem cerdas yang dapat berinteraksi dengan pengguna, memproses dan menganalisis data teks, serta menghasilkan keluaran yang bermakna.

Dengan memasukkan LLM ke dalam arsitektur aplikasi, Anda dapat membuat komponen AI yang memahami dan memproses masukan pengguna, menghasilkan konten dinamis, dan memberikan rekomendasi atau tindakan cerdas. Namun bekerja dengan LLM memerlukan pertimbangan cermat terhadap kebutuhan sumber daya dan trade-off kinerja. LLM membutuhkan komputasi intensif dan mungkin memerlukan daya pemrosesan dan memori yang signifikan (dengan kata lain, uang) untuk beroperasi. Kebanyakan dari kita perlu menilai implikasi biaya dari mengintegrasikan LLM ke dalam aplikasi kita dan bertindak sesuai dengan hal tersebut.

Memahami Inferensi

Inferensi mengacu pada proses di mana sebuah model menghasilkan prediksi atau keluaran berdasarkan data baru yang belum pernah dilihat sebelumnya. Ini adalah fase di mana model yang telah dilatih digunakan untuk membuat keputusan atau menghasilkan teks, gambar, atau konten lainnya sebagai respons terhadap masukan pengguna.

Selama fase pelatihan, model AI belajar dari kumpulan data yang besar dengan menyesuaikan parameter-parameternya untuk meminimalkan kesalahan dalam prediksinya.

Setelah dilatih, model dapat menerapkan apa yang telah dipelajarinya pada data baru. Inferensi adalah cara model menggunakan pola dan pengetahuan yang telah dipelajarinya untuk menghasilkan keluaran.

Untuk LLM, inferensi melibatkan pengambilan prompt atau teks masukan dan menghasilkan respons yang koheren dan relevan secara kontekstual, sebagai aliran *token* (yang akan kita bahas segera). Ini bisa berupa menjawab pertanyaan, melengkapi kalimat, menghasilkan cerita, atau menerjemahkan teks, di antara banyak tugas lainnya.



Berbeda dengan cara kita berpikir, “pemikiran” model AI melalui inferensi terjadi dalam satu operasi tanpa status. Artinya, pemikirannya terbatas pada proses generasinya. Ia secara harfiah harus berpikir dengan keras, seolah-olah saya mengajukan pertanyaan kepada Anda dan hanya menerima respons dari Anda dalam gaya “aliran kesadaran”.

Model Bahasa Besar Hadir dalam Berbagai Ukuran dan Variasi

Meskipun hampir semua model bahasa besar (LLM) populer didasarkan pada arsitektur transformer yang sama dan dilatih pada kumpulan data teks yang sangat besar, mereka hadir dalam berbagai ukuran dan disetel halus untuk tujuan yang berbeda. Ukuran LLM, yang diukur dari jumlah parameter dalam jaringan sarafnya, memiliki dampak besar pada kemampuannya. Model yang lebih besar dengan lebih banyak parameter, seperti GPT-4, yang dikabarkan memiliki 1 hingga 2 triliun parameter, umumnya lebih berpengetahuan dan mampu dibandingkan model yang lebih kecil. Namun, model yang lebih besar juga membutuhkan daya komputasi yang jauh lebih besar untuk dijalankan, yang berarti biaya lebih tinggi ketika Anda menggunakannya melalui panggilan API.

Untuk membuat LLM lebih praktis dan disesuaikan untuk kasus penggunaan tertentu, model dasar sering disetel halus pada kumpulan data yang lebih terarah. Misalnya, LLM dapat dilatih pada korpus dialog yang besar untuk mengkhususkannya pada AI

konversasional. Yang lain dilatih pada kode untuk membekalinya dengan pengetahuan pemrograman. Bahkan ada model yang dilatih khusus untuk interaksi gaya bermain peran dengan pengguna!

Model Pengambilan vs Model Generatif

Dalam dunia model bahasa besar (LLM), terdapat dua pendekatan utama untuk menghasilkan respons: model berbasis pengambilan dan model generatif. Setiap pendekatan memiliki kekuatan dan kelemahannya masing-masing, dan memahami perbedaan di antara keduanya dapat membantu Anda memilih model yang tepat untuk kasus penggunaan spesifik Anda.

Model Berbasis Pengambilan

Model berbasis pengambilan, yang juga dikenal sebagai model pengambilan informasi, menghasilkan respons dengan mencari dalam basis data besar berisi teks yang sudah ada dan memilih bagian yang paling relevan berdasarkan kueri masukan. Model-model ini tidak menghasilkan teks baru dari awal tetapi lebih kepada menggabungkan kutipan dari basis data untuk membentuk respons yang koheren.

Salah satu keunggulan utama model berbasis pengambilan adalah kemampuannya untuk memberikan informasi yang akurat secara faktual dan terkini. Karena mereka mengandalkan basis data teks yang telah dikurasi, mereka dapat mengambil informasi yang relevan dari sumber-sumber terpercaya dan menyajikannya kepada pengguna. Hal ini membuat mereka sangat cocok untuk aplikasi yang membutuhkan jawaban yang tepat dan faktual, seperti sistem tanya-jawab atau basis pengetahuan.

Namun, model berbasis pengambilan memiliki beberapa keterbatasan. Kualitas mereka hanya sebaik basis data yang mereka cari, sehingga kualitas dan cakupan basis data secara langsung memengaruhi kinerja model. Selain itu, model-model ini mungkin kesulitan menghasilkan respons yang koheren dan terdengar alami, karena mereka terbatas pada teks yang tersedia dalam basis data.

Kami tidak membahas penggunaan model pengambilan murni dalam buku ini.

Model Generatif

Model generatif, di sisi lain, menciptakan teks baru dari awal berdasarkan pola dan hubungan yang mereka pelajari selama pelatihan. Model-model ini menggunakan pemahaman mereka tentang bahasa untuk menghasilkan respons baru yang disesuaikan dengan prompt masukan.

Kekuatan utama model generatif adalah kemampuan mereka untuk menghasilkan teks yang kreatif, koheren, dan relevan secara kontekstual. Mereka dapat terlibat dalam percakapan terbuka, menghasilkan cerita, dan bahkan menulis kode. Hal ini membuat mereka ideal untuk aplikasi yang membutuhkan interaksi yang lebih terbuka dan dinamis, seperti chatbot, pembuatan konten, dan asisten penulisan kreatif.

Namun, model generatif terkadang dapat menghasilkan informasi yang tidak konsisten atau salah secara faktual, karena mereka mengandalkan pola yang dipelajari selama pelatihan daripada basis data fakta yang telah dikurasi. Mereka juga mungkin lebih rentan terhadap bias dan halusinasi, menghasilkan teks yang masuk akal tetapi tidak selalu benar.

Contoh LLM generatif termasuk seri GPT dari OpenAI (GPT-3, GPT-4) dan Claude dari Anthropic.

Model Hibrida

Beberapa LLM yang tersedia secara komersial menggabungkan pendekatan pengambilan dan generatif dalam model hibrida. Model-model ini menggunakan teknik pengambilan untuk menemukan informasi yang relevan dari database dan kemudian menggunakan teknik generatif untuk menyintesis informasi tersebut menjadi respons yang koheren.

Model hibrida bertujuan untuk menggabungkan akurasi faktual dari model berbasis pengambilan dengan kemampuan generasi bahasa natural dari model generatif. Model-model ini dapat memberikan informasi yang lebih dapat diandalkan dan terkini sambil tetap mempertahankan kemampuan untuk terlibat dalam percakapan yang bersifat terbuka.

Ketika memilih antara model berbasis pengambilan dan model generatif, Anda harus mempertimbangkan persyaratan khusus aplikasi Anda. Jika tujuan utamanya adalah memberikan informasi faktual yang akurat, model berbasis pengambilan mungkin menjadi pilihan terbaik. Jika aplikasi membutuhkan interaksi yang lebih terbuka dan kreatif, model generatif mungkin lebih cocok. Model hibrida menawarkan keseimbangan antara kedua pendekatan tersebut dan bisa menjadi pilihan yang baik untuk aplikasi yang membutuhkan baik akurasi faktual maupun generasi bahasa natural.

Pada akhirnya, pilihan antara model berbasis pengambilan dan model generatif bergantung pada kasus penggunaan spesifik dan pertimbangan antara akurasi, kreativitas, dan fleksibilitas. Dengan memahami kekuatan dan keterbatasan setiap pendekatan, Anda dapat membuat keputusan yang tepat.

Penyetelan Instruksi

Penyetelan instruksi adalah subset dari pelatihan percakapan yang menggunakan prompt dan respons yang ditulis manusia untuk membuat LLM lebih baik dalam memahami dan mengikuti instruksi. Berikut beberapa contoh sederhana untuk mengilustrasikan bagaimana penyetelan instruksi pada prompt dan respons yang ditulis manusia membuat LLM lebih baik dalam memahami dan mengikuti instruksi:

Contoh 1: Layanan Pelanggan

Instruksi: “Berikan respons untuk pelanggan yang meminta pengembalian dana.”

Prompt yang Ditulis Manusia: “Saya ingin mengajukan permintaan pengembalian dana untuk pembelian saya baru-baru ini. Produknya tidak sesuai dengan harapan saya.”

Respons yang Ditulis Manusia: “Mohon maaf mendengar bahwa produk tersebut tidak sesuai dengan harapan Anda. Untuk memproses pengembalian dana Anda, mohon berikan nomor pesanan dan alasan pengembalian. Kami akan meninjau permintaan Anda dan menghubungi Anda kembali dalam waktu 2-3 hari kerja.”

Contoh 2: Bantuan Pendidikan

Instruksi: “Jelaskan konsep fotosintesis kepada anak berusia 10 tahun.”

Prompt Buatan Manusia: “Bisakah kamu menjelaskan bagaimana tumbuhan membuat makanannya?”

Respons Buatan Manusia: “Tentu! Tumbuhan membuat makanannya melalui proses yang disebut fotosintesis. Mereka menggunakan sinar matahari, air, dan karbon dioksida untuk menciptakan makanan dan oksigen. Daun tumbuhan memiliki bagian

khusus yang disebut klorofil yang membantu mereka menangkap sinar matahari.”

Contoh 3: Asisten Pribadi

Instruksi: “Jadwalkan rapat untuk hari Selasa depan pukul 3 sore.”

Prompt Buatan Manusia: “Bisakah kamu mengatur rapat untuk hari Selasa pukul 3 sore?”

Respons Buatan Manusia: “Tentu, saya telah menjadwalkan rapat untuk Anda hari Selasa depan pukul 3 sore. Apakah ada hal lain yang Anda butuhkan?”

Hasilnya adalah ekosistem LLM yang beragam dengan ukuran berbeda dan spesialisasi yang berbeda. Model yang lebih kecil dengan rentang 1-7 miliar parameter menyediakan kemampuan bahasa umum yang baik sambil tetap efisien untuk dijalankan.

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

Model berukuran menengah sekitar 30-70 miliar parameter menawarkan kemampuan penalaran dan mengikuti instruksi yang lebih kuat.

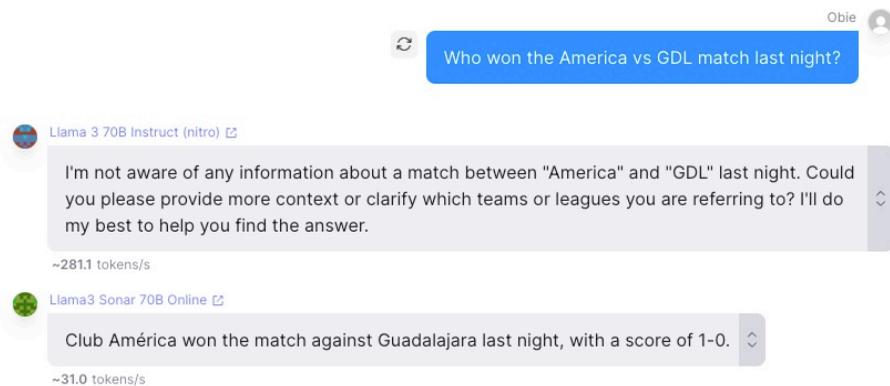
- Llama 3 70B
- Qwen2 70B
- Mixtral 8x22B

Ketika memilih LLM untuk diintegrasikan ke dalam aplikasi, Anda harus menyeimbangkan kemampuan model dengan faktor-faktor praktis seperti biaya, latensi, panjang konteks, dan penyaringan konten. Model yang lebih kecil yang dilatih dengan instruksi seringkali menjadi pilihan terbaik untuk tugas bahasa yang lebih sederhana, sementara

model terbesar mungkin diperlukan untuk penalaran atau analisis yang kompleks. Data pelatihan model juga merupakan pertimbangan penting, karena ini menentukan tanggal batas pengetahuan model.



Model tertentu, seperti beberapa dari Perplexity terhubung ke sumber informasi waktu nyata, sehingga mereka secara efektif tidak memiliki tanggal batas. Ketika Anda mengajukan pertanyaan kepada mereka, mereka dapat secara mandiri memutuskan untuk melakukan pencarian web dan mengambil halaman web tertentu untuk menghasilkan jawaban.

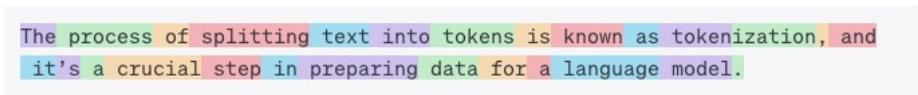


Gambar 1. Llama3 dengan dan tanpa akses online

Pada akhirnya, tidak ada LLM yang cocok untuk segala keperluan. Memahami variasi dalam ukuran model, arsitektur, dan pelatihan adalah kunci untuk memilih model yang tepat untuk kasus penggunaan tertentu. Bereksperimen dengan berbagai model adalah satu-satunya cara praktis untuk mengetahui mana yang memberikan kinerja terbaik untuk tugas yang sedang dihadapi.

Tokenisasi: Memecah Teks Menjadi Bagian-Bagian

Sebelum model bahasa besar dapat memproses teks, teks tersebut perlu dipecah menjadi unit-unit yang lebih kecil yang disebut *token*. Token bisa berupa kata-kata individual, bagian dari kata, atau bahkan karakter tunggal. Proses pemecahan teks menjadi token dikenal sebagai tokenisasi, dan ini adalah langkah penting dalam mempersiapkan data untuk model bahasa.



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

Gambar 2. Kalimat ini mengandung 27 token

LLM yang berbeda menggunakan strategi tokenisasi yang berbeda, yang dapat memiliki dampak signifikan pada kinerja dan kemampuan model. Beberapa penokenisasi umum yang digunakan oleh LLM meliputi:

- **GPT (Byte Pair Encoding/Pengkodean Pasangan Byte):** Penokenisasi GPT menggunakan teknik yang disebut byte pair encoding (BPE) untuk memecah teks menjadi unit-unit sub-kata. BPE secara berulang menggabungkan pasangan byte yang paling sering muncul dalam korpus teks, membentuk kosakata token sub-kata. Ini memungkinkan penokenisasi untuk menangani kata-kata yang jarang dan baru dengan memecahnya menjadi bagian-bagian sub-kata yang lebih umum. Penokenisasi GPT digunakan oleh model seperti GPT-3 dan GPT-4.
- **Llama (SentencePiece):** Penokenisasi Llama menggunakan pustaka SentencePiece, yang merupakan penokenisasi dan detokenisasi teks tanpa pengawasan. SentencePiece memperlakukan teks masukan sebagai urutan karakter Unicode dan mempelajari kosakata sub-kata berdasarkan korpus pelatihan. Ini dapat menangani bahasa apa pun yang dapat dikodekan dalam Unicode, membuatnya sangat cocok untuk model multibahasa. Penokenisasi Llama digunakan oleh model seperti Llama dari Meta dan Alpaca.

- **SentencePiece (Unigram):** Tokenizer SentencePiece juga dapat menggunakan algoritma berbeda yang disebut Unigram, yang didasarkan pada teknik regularisasi subkata. Tokenisasi Unigram menentukan kosakata subkata optimal berdasarkan model bahasa unigram, yang memberikan probabilitas ke unit-unit subkata individual. Pendekatan ini dapat menghasilkan subkata yang lebih bermakna secara semantik dibandingkan dengan BPE. SentencePiece dengan Unigram digunakan oleh model-model seperti T5 dari Google dan BERT.
- **Google Gemini (Tokenisasi Multimodal):** Google Gemini menggunakan skema tokenisasi yang dirancang untuk menangani berbagai jenis data, termasuk teks, gambar, audio, video, dan kode. Kemampuan multimodal ini memungkinkan Gemini untuk memproses dan mengintegrasikan berbagai bentuk informasi. Yang perlu dicatat, Google Gemini 1.5 Pro memiliki jendela konteks yang dapat menangani jutaan token, jauh lebih besar dari model-model sebelumnya. Jendela konteks yang luas ini memungkinkan model untuk memproses konteks yang lebih besar, yang berpotensi menghasilkan respons yang lebih akurat. Namun, penting untuk dicatat bahwa skema tokenisasi Gemini jauh lebih mendekati satu token per karakter dibandingkan model lainnya. Ini berarti biaya sebenarnya untuk menggunakan model Gemini bisa jauh lebih tinggi dari yang diharapkan jika Anda terbiasa menggunakan model seperti GPT, karena harga Google didasarkan pada karakter daripada token.

Pilihan tokenizer mempengaruhi beberapa aspek LLM, termasuk:

- **Ukuran kosakata:** Tokenizer menentukan ukuran kosakata model, yang merupakan kumpulan token unik yang dikenalnya. Kosakata yang lebih besar dan lebih terperinci dapat membantu model menangani berbagai kata dan frasa yang lebih luas dan bahkan menjadi multi-modal (mampu memahami dan menghasilkan lebih dari sekadar teks), tetapi juga meningkatkan kebutuhan memori dan kompleksitas komputasi model.

- **Penanganan kata-kata langka dan tidak dikenal:** Tokenizer yang menggunakan unit subkata, seperti BPE dan SentencePiece, dapat memecah kata-kata langka dan tidak dikenal menjadi bagian-bagian subkata yang lebih umum. Ini memungkinkan model untuk membuat perkiraan terdidik tentang arti kata-kata yang belum pernah dilihatnya sebelumnya, berdasarkan subkata yang dikandungnya.
- **Dukungan multibahasa:** Tokenizer seperti SentencePiece, yang dapat menangani bahasa apapun yang dapat dikodekan Unicode, sangat cocok untuk model multibahasa yang perlu memproses teks dalam berbagai bahasa.

Ketika memilih LLM untuk aplikasi tertentu, penting untuk mempertimbangkan tokenizer yang digunakan dan seberapa baik kesesuaiannya dengan kebutuhan pemrosesan bahasa spesifik dari tugas yang dihadapi. Tokenizer dapat memiliki dampak signifikan pada kemampuan model dalam menangani terminologi khusus domain, kata-kata langka, dan teks multibahasa.

Ukuran Konteks: Berapa Banyak Informasi yang Dapat Digunakan Model Bahasa Selama Inferensi?

Ketika membahas model bahasa, ukuran konteks mengacu pada jumlah teks yang dapat dipertimbangkan oleh model saat memproses atau menghasilkan responnya. Ini pada dasarnya adalah ukuran seberapa banyak informasi yang dapat “diingat” model dan digunakan untuk menginformasikan keluarannya (dinyatakan dalam token). Ukuran konteks dari model bahasa dapat memiliki dampak signifikan pada kemampuannya dan jenis tugas yang dapat dilakukannya secara efektif.

Apa itu Ukuran Konteks?

Secara teknis, ukuran konteks ditentukan oleh jumlah token (kata atau potongan kata) yang dapat diproses model bahasa dalam satu rangkaian input. Ini sering disebut sebagai “rentang perhatian” atau “jendela konteks” model. Semakin besar ukuran konteks,

semakin banyak teks yang dapat dipertimbangkan model sekaligus ketika menghasilkan respons atau melakukan tugas.

Model bahasa yang berbeda memiliki ukuran konteks yang bervariasi, mulai dari beberapa ratus token hingga jutaan token. Sebagai referensi, paragraf teks biasa mungkin berisi sekitar 100-150 token, sementara seluruh buku bisa berisi puluhan atau ratusan ribu token.

Bahkan ada penelitian tentang metode efisien untuk mengembangkan Model Bahasa Besar (LLM) berbasis Transformer untuk **input yang tak terbatas** dengan memori dan komputasi yang terbatas.

Mengapa Ukuran Konteks Penting?

Ukuran konteks dari model bahasa memiliki dampak signifikan pada kemampuannya untuk memahami dan menghasilkan teks yang koheren dan relevan secara kontekstual. Berikut beberapa alasan utama mengapa ukuran konteks penting:

1. **Memahami konten panjang:** Model dengan ukuran konteks yang lebih besar dapat lebih baik dalam memahami dan menganalisis teks yang lebih panjang, seperti artikel, laporan, atau bahkan buku lengkap. Ini sangat penting untuk tugas-tugas seperti peringkasan dokumen, tanya jawab, dan analisis konten.
2. **Menjaga koherensi:** Jendela konteks yang lebih besar memungkinkan model untuk mempertahankan koherensi dan konsistensi di sepanjang keluaran yang lebih panjang. Ini penting untuk tugas-tugas seperti pembuatan cerita, sistem dialog, dan pembuatan konten, di mana mempertahankan narasi atau topik yang konsisten sangat penting. Ini juga sangat penting ketika menggunakan LLM untuk menghasilkan atau mentransformasi data terstruktur.

3. **Menangkap ketergantungan jarak jauh:** Beberapa tugas bahasa membutuhkan pemahaman hubungan antara kata atau frasa yang terpisah jauh dalam sebuah teks. Model dengan ukuran konteks yang lebih besar lebih mampu menangkap ketergantungan jarak jauh ini, yang dapat penting untuk tugas-tugas seperti analisis sentimen, penerjemahan, dan pemahaman bahasa.
4. **Menangani instruksi kompleks:** Dalam aplikasi di mana model bahasa digunakan untuk mengikuti instruksi kompleks dan bertahap, ukuran konteks yang lebih besar memungkinkan model untuk mempertimbangkan seluruh rangkaian instruksi ketika menghasilkan respons, bukan hanya beberapa kata terakhir.

Contoh Model Bahasa dengan Ukuran Konteks Berbeda

Berikut beberapa contoh model bahasa dengan ukuran konteks yang berbeda:

- OpenAI GPT-3.5 Turbo: 4.095 token
- Mistral 7B Instruct: 32.768 token
- Anthropic Claude v1: 100.000 token
- OpenAI GPT-4 Turbo: 128.000 token
- Anthropic Claude v2: 200.000 token
- Google Gemini Pro 1.5: 2,8 juta token

Seperti yang dapat Anda lihat, terdapat rentang ukuran konteks yang luas di antara model-model ini, mulai dari sekitar 4.000 token untuk model OpenAI GPT-3.5 Turbo hingga 200.000 token untuk model Anthropic Claude v2. Beberapa model, seperti Google PaLM 2 dan OpenAI GPT-4, menawarkan varian berbeda dengan ukuran konteks yang lebih besar (misalnya, versi “32k”), yang dapat menangani urutan input yang lebih panjang. Dan saat ini (April 2024) Google Gemini Pro membanggakan hampir 3 juta token!

Perlu dicatat bahwa ukuran konteks dapat bervariasi tergantung pada implementasi dan versi spesifik dari model tertentu. Misalnya, model OpenAI GPT-4 original memiliki

ukuran konteks 8.191 token, sementara varian GPT-4 yang lebih baru seperti Turbo dan 4o memiliki ukuran konteks yang jauh lebih besar yaitu 128.000 token.

Sam Altman telah membandingkan keterbatasan konteks saat ini dengan kilobyte memori kerja yang harus dihadapi programmer komputer pribadi di era 80-an, dan mengatakan bahwa dalam waktu dekat kita akan dapat memasukkan “semua data pribadi Anda” ke dalam konteks sebuah model bahasa besar.

Memilih Ukuran Konteks yang Tepat

Saat memilih model bahasa untuk aplikasi tertentu, penting untuk mempertimbangkan kebutuhan ukuran konteks dari tugas yang akan dikerjakan. Untuk tugas yang melibatkan potongan teks pendek dan terpisah, seperti analisis sentimen atau tanya jawab sederhana, ukuran konteks yang lebih kecil mungkin sudah mencukupi. Namun, untuk tugas yang membutuhkan pemahaman dan pembuatan teks yang lebih panjang dan kompleks, ukuran konteks yang lebih besar kemungkinan akan diperlukan.

Perlu diketahui bahwa ukuran konteks yang lebih besar seringkali disertai dengan peningkatan biaya komputasi dan waktu pemrosesan yang lebih lambat, karena model perlu mempertimbangkan lebih banyak informasi saat menghasilkan respons. Dengan demikian, Anda harus menyeimbangkan antara ukuran konteks dan kinerja saat memilih model bahasa untuk aplikasi Anda.

Mengapa tidak langsung memilih model dengan ukuran konteks terbesar dan mengisinya dengan sebanyak mungkin informasi? Yah, selain faktor kinerja, pertimbangan utama lainnya adalah biaya. Pada Maret 2024, satu siklus prompt-respons menggunakan Google Gemini Pro 1.5 dengan konteks penuh akan

menghabiskan biaya hampir \$8 (USD). Jika Anda memiliki kasus penggunaan yang membenarkan biaya tersebut, silakan saja! Tapi untuk kebanyakan aplikasi, biayanya terlalu mahal hingga beberapa kali lipat.

Mencari Jarum dalam Tumpukan Jerami

Konsep mencari jarum dalam tumpukan jerami telah lama menjadi metafora untuk tantangan pengambilan data dalam kumpulan data besar. Dalam ranah LLM, kita sedikit memodifikasi analogi ini. Bayangkan kita tidak hanya mencari satu fakta yang terpendam dalam teks yang sangat besar (seperti antologi lengkap esai Paul Graham), tetapi beberapa fakta yang tersebar di dalamnya. Skenario ini lebih mirip dengan mencari beberapa jarum di lapangan yang luas, bukan hanya satu tumpukan jerami. Inilah yang menarik: kita tidak hanya perlu menemukan jarum-jarum ini, tetapi juga harus merajutnya menjadi benang yang koheren.

Ketika bertugas mengambil dan menalar tentang berbagai fakta yang tertanam dalam konteks panjang, LLM menghadapi tantangan ganda. Pertama, ada masalah akurasi pengambilan yang sederhana—secara alami menurun seiring bertambahnya jumlah fakta. Ini dapat dimaklumi; bagaimanapun juga, melacak berbagai detail di seluruh teks yang luas membebani bahkan model yang paling canggih sekalipun.

Kedua, dan mungkin lebih kritis lagi, adalah tantangan dalam melakukan penalaran dengan fakta-fakta tersebut. Mengidentifikasi fakta adalah satu hal; namun mensintesiskannya menjadi narasi atau jawaban yang koheren adalah hal yang berbeda. Di sinilah ujian yang sesungguhnya muncul. Kinerja LLM dalam tugas-tugas penalaran cenderung mengalami penurunan lebih besar dibandingkan dalam tugas-tugas pengambilan informasi sederhana. Penurunan ini bukan hanya tentang volume; ini tentang interaksi rumit antara konteks, relevansi, dan inferensi.

Mengapa hal ini terjadi? Mari kita perhatikan dinamika memori dan atensi dalam

kognisi manusia, yang sampai taraf tertentu tercermin dalam LLM. Ketika memproses informasi dalam jumlah besar, LLM, seperti halnya manusia, dapat kehilangan jejak detail-detail awal saat menyerap informasi baru. Hal ini terutama benar pada model-model yang tidak secara khusus dirancang untuk memprioritaskan atau meninjau ulang segmen-segmen teks sebelumnya secara otomatis.

Lebih lanjut, kemampuan LLM untuk merangkai fakta-fakta yang diambil menjadi respons yang koheren mirip dengan proses pembangunan narasi. Ini membutuhkan bukan hanya pengambilan informasi tetapi juga pemahaman mendalam dan penempatan kontekstual, yang masih menjadi tantangan berat bagi AI saat ini.

Jadi, apa artinya ini bagi kita sebagai pengembang dan integrator teknologi-teknologi ini? Kita perlu sangat memahami keterbatasan ini ketika merancang sistem yang mengandalkan LLM untuk menangani tugas-tugas kompleks dan panjang. Memahami bahwa kinerja mungkin menurun dalam kondisi tertentu membantu kita menetapkan ekspektasi yang realistis dan merancang mekanisme cadangan atau strategi pendukung yang lebih baik.

Modalitas: Melampaui Teks

Sementara mayoritas model bahasa saat ini berfokus pada pemrosesan dan penghasilan teks, terdapat tren yang berkembang menuju model multimodal yang secara alami dapat menerima dan menghasilkan berbagai jenis data, seperti gambar, audio, dan video. Model-model multimodal ini membuka kemungkinan baru untuk aplikasi berbasis AI yang dapat memahami dan menghasilkan konten di berbagai modalitas.

Apa itu Modalitas?

Dalam konteks model bahasa, modalitas mengacu pada berbagai jenis data yang dapat diproses dan dihasilkan oleh sebuah model. Modalitas yang paling umum adalah teks, yang mencakup bahasa tertulis dalam berbagai bentuk seperti buku, artikel, situs web,

dan postingan media sosial. Namun, ada beberapa modalitas lain yang semakin banyak diintegrasikan ke dalam model bahasa:

- **Gambar:** Data visual seperti foto, ilustrasi, dan diagram.
- **Audio:** Data suara seperti ucapan, musik, dan suara lingkungan.
- **Video:** Data visual bergerak, seringkali disertai audio, seperti klip video dan film.

Setiap modalitas menghadirkan tantangan dan peluang yang unik bagi model bahasa. Sebagai contoh, gambar mengharuskan model untuk memahami konsep dan hubungan visual, sementara audio mengharuskan model untuk memproses dan menghasilkan ucapan dan suara lainnya.

Model Bahasa Multimodal

Model bahasa multimodal dirancang untuk menangani berbagai modalitas dalam satu model. Model-model ini biasanya memiliki komponen atau lapisan khusus yang dapat memahami input dan menghasilkan data output dalam modalitas yang berbeda. Beberapa contoh penting model bahasa multimodal meliputi:

- **OpenAI's GPT-4o:** GPT-4o adalah model bahasa besar yang secara alami memahami dan memproses audio ucapan selain teks. Kemampuan ini memungkinkan GPT-4o untuk melakukan tugas-tugas seperti mentranskripsikan bahasa lisan, menghasilkan teks dari input audio, dan memberikan respons berdasarkan pertanyaan lisan.
- **OpenAI's GPT-4 dengan input visual:** GPT-4 adalah model bahasa besar yang dapat memproses teks dan gambar. Ketika diberikan gambar sebagai input, GPT-4 dapat menganalisis isi gambar dan menghasilkan teks yang mendeskripsikan atau merespons informasi visual tersebut.
- **Google's Gemini:** Gemini adalah model multimodal yang dapat menangani teks, gambar, dan video. Model ini menggunakan arsitektur terpadu yang

memungkinkan pemahaman dan generasi lintas modal, memungkinkan tugas-tugas seperti pemberian keterangan gambar, peringkasan video, dan penjawaban pertanyaan visual.

- **DALL-E dan Stable Diffusion:** Meskipun bukan model bahasa dalam pengertian tradisional, model-model ini menunjukkan kekuatan AI multimodal dengan menghasilkan gambar dari deskripsi tekstual. Mereka memperlihatkan potensi model yang dapat menerjemahkan antara modalitas yang berbeda.

Manfaat dan Aplikasi Model Multimodal

Model bahasa multimodal menawarkan beberapa manfaat dan memungkinkan berbagai aplikasi, termasuk:

- **Pemahaman yang lebih baik:** Dengan memproses informasi dari berbagai modalitas, model-model ini dapat memperoleh pemahaman yang lebih komprehensif tentang dunia, mirip dengan cara manusia belajar dari berbagai input sensorik.
- **Generasi lintas modal:** Model multimodal dapat menghasilkan konten dalam satu modalitas berdasarkan input dari modalitas lain, seperti membuat gambar dari deskripsi teks atau menghasilkan ringkasan video dari artikel tertulis.
- **Aksesibilitas:** Model multimodal dapat membuat informasi lebih mudah diakses dengan menerjemahkan antar modalitas, seperti menghasilkan deskripsi teks dari gambar untuk pengguna dengan gangguan penglihatan atau membuat versi audio dari konten tertulis.
- **Aplikasi kreatif:** Model multimodal dapat digunakan untuk tugas-tugas kreatif seperti menghasilkan seni, musik, atau video berdasarkan prompt teks, membuka kemungkinan baru bagi seniman dan kreator konten.

Seiring dengan kemajuan model bahasa multimodal, mereka kemungkinan akan memainkan peran yang semakin penting dalam pengembangan aplikasi berbasis AI yang dapat memahami dan menghasilkan konten di berbagai modalitas. Hal ini akan memungkinkan interaksi yang lebih alami dan intuitif antara manusia dan sistem AI, serta membuka kemungkinan baru untuk ekspresi kreatif dan penyebaran pengetahuan.

Ekosistem Penyedia

Ketika berbicara tentang mengintegrasikan model bahasa besar (LLM) ke dalam aplikasi, Anda memiliki berbagai pilihan yang terus berkembang. Setiap penyedia LLM utama, seperti OpenAI, Anthropic, Google, dan Cohere, menawarkan ekosistem modelnya sendiri, API, dan perangkat. Memilih penyedia yang tepat melibatkan pertimbangan berbagai faktor, termasuk harga, kinerja, penyaringan konten, privasi data, dan opsi penyesuaian.

OpenAI

OpenAI adalah salah satu penyedia LLM yang paling terkenal, dengan seri GPT-nya (GPT-3, GPT-4) yang banyak digunakan dalam berbagai aplikasi. OpenAI menawarkan API yang mudah digunakan yang memungkinkan Anda mengintegrasikan model mereka ke dalam aplikasi dengan mudah. Mereka menyediakan berbagai model dengan kemampuan dan harga yang berbeda, mulai dari model Ada tingkat pemula hingga model Davinci yang kuat.

Ekosistem OpenAI juga mencakup perangkat seperti OpenAI Playground, yang memungkinkan Anda bereksperimen dengan prompt dan melakukan penalaan model untuk kasus penggunaan tertentu. Mereka menawarkan opsi penyaringan konten untuk membantu mencegah pembuatan konten yang tidak pantas atau berbahaya.

Ketika menggunakan model OpenAI secara langsung, saya mengandalkan pustaka [ruby-openai](#) karya Alex Rudall.

Anthropic

Anthropic adalah pemain utama lainnya dalam bidang LLM, dengan model Claude mereka yang semakin populer karena kinerja yang kuat dan pertimbangan etis. Anthropic fokus pada pengembangan sistem AI yang aman dan bertanggung jawab, dengan penekanan kuat pada penyaringan konten dan menghindari output yang berbahaya.

Ekosistem Anthropic mencakup API Claude, yang memungkinkan Anda mengintegrasikan model ke dalam aplikasi mereka, serta perangkat untuk rekayasa prompt dan penalaan. Mereka juga menawarkan model Claude Instant, yang menggabungkan kemampuan pencarian web untuk respons yang lebih mutakhir dan faktual.

Ketika menggunakan model Anthropic secara langsung, saya mengandalkan pustaka [anthropic](#) karya Alex Rudall.

Google

Google telah mengembangkan beberapa LLM yang kuat, termasuk Gemini, BERT, T5, dan PaLM. Model-model ini dikenal karena kinerja yang kuat dalam berbagai tugas pemrosesan bahasa alami. Ekosistem Google mencakup pustaka TensorFlow dan Keras, yang menyediakan alat dan kerangka kerja untuk membangun dan melatih model pembelajaran mesin.

Google juga menawarkan Cloud AI Platform, yang memungkinkan Anda dengan mudah menerapkan dan menskalakan model mereka di cloud. Mereka menyediakan berbagai model yang sudah dilatih sebelumnya dan API untuk tugas-tugas seperti analisis sentimen, pengenalan entitas, dan terjemahan.

Meta

Meta, yang sebelumnya dikenal sebagai Facebook, sangat berkomitmen dalam pengembangan model bahasa besar, yang ditonjolkan melalui peluncuran model-model seperti LLaMA dan OPT. Model-model ini menonjol karena kinerja yang kuat dalam berbagai

tugas bahasa dan sebagian besar tersedia melalui saluran sumber terbuka, mendukung komitmen Meta terhadap penelitian dan kolaborasi komunitas.

Ekosistem Meta terutama dibangun di sekitar PyTorch, sebuah pustaka pembelajaran mesin sumber terbuka yang disukai karena kemampuan komputasi dinamis dan fleksibilitasnya, memfasilitasi penelitian dan pengembangan AI yang inovatif.

Selain penawaran teknisnya, Meta memberikan penekanan kuat pada pengembangan AI yang etis. Mereka menerapkan penyaringan konten yang kuat dan fokus pada pengurangan bias, sejalan dengan tujuan yang lebih luas mereka untuk keamanan dan tanggung jawab dalam aplikasi AI.

Cohere

Cohere adalah pendatang baru dalam ruang LLM, yang berfokus untuk membuat LLM lebih mudah diakses dan digunakan dibandingkan pesaing. Ekosistem mereka mencakup Cohere API, yang menyediakan akses ke berbagai model yang sudah dilatih sebelumnya untuk tugas-tugas seperti pembuatan teks, klasifikasi, dan peringkasan.

Cohere juga menawarkan alat untuk rekayasa prompt, penyetulan halus, dan penyaringan konten. Mereka menekankan privasi dan keamanan data, dengan fitur seperti penyimpanan data terenkripsi dan kontrol akses.

Ollama

Ollama adalah platform yang dihosting sendiri yang memungkinkan pengguna mengelola dan menerapkan berbagai model bahasa besar (LLM) secara lokal di mesin mereka, memberikan mereka kendali penuh atas model AI mereka tanpa bergantung pada layanan cloud eksternal. Pengaturan ini ideal bagi mereka yang memprioritaskan privasi data dan ingin menangani operasi AI mereka secara internal.

Platform ini mendukung berbagai model, termasuk versi-versi Llama, Phi, Gemma, dan Mistral, yang bervariasi dalam ukuran dan kebutuhan komputasinya. Ollama

memudahkan pengunduhan dan pengoperasian model-model ini langsung dari baris perintah menggunakan perintah sederhana seperti `ollama run <model_name>`, dan dirancang untuk bekerja di berbagai sistem operasi termasuk macOS, Linux, dan Windows.

Bagi pengembang yang ingin mengintegrasikan model sumber terbuka ke dalam aplikasi mereka tanpa menggunakan API jarak jauh, Ollama menawarkan CLI untuk mengelola siklus hidup model mirip dengan alat pengelolaan kontainer. Ini juga mendukung konfigurasi dan prompt kustom, memungkinkan tingkat kustomisasi yang tinggi untuk menyesuaikan model dengan kebutuhan atau kasus penggunaan tertentu.

Ollama sangat cocok untuk pengguna yang mahir teknologi dan pengembang karena antarmuka baris perintahnya dan fleksibilitas yang ditawarkan dalam mengelola dan menerapkan model AI. Ini menjadikannya alat yang ampuh untuk bisnis dan individu yang membutuhkan kemampuan AI yang kuat tanpa mengorbankan keamanan dan kontrol.

Platform Multi-Model

Selain itu, ada penyedia yang menampung berbagai model sumber terbuka, seperti Together.ai dan Groq. Platform-platform ini menawarkan fleksibilitas dan kustomisasi, memungkinkan Anda menjalankan dan, dalam beberapa kasus, bahkan melakukan penyyetelan halus model sumber terbuka sesuai dengan kebutuhan spesifik Anda. Sebagai contoh, Together.ai menyediakan akses ke berbagai LLM sumber terbuka, memungkinkan pengguna bereksperimen dengan berbagai model dan konfigurasi. Groq berfokus pada penyediaan penyelesaian berkinerja ultra tinggi yang pada saat penulisan buku ini tampak hampir ajaib

Memilih Penyedia LLM

Saat memilih penyedia LLM, Anda harus mempertimbangkan faktor-faktor seperti:

- **Harga:** Penyedia yang berbeda menawarkan model penetapan harga yang berbeda, mulai dari bayar per penggunaan hingga rencana berbasis langganan. Penting untuk mempertimbangkan perkiraan penggunaan dan anggaran saat memilih penyedia.
- **Performa:** Performa LLM dapat sangat bervariasi antar penyedia, jadi penting untuk melakukan pengujian performa dan menguji model pada kasus penggunaan tertentu sebelum membuat keputusan.
- **Penyaringan Konten:** Tergantung pada aplikasi, penyaringan konten mungkin menjadi pertimbangan penting. Beberapa penyedia menawarkan opsi penyaringan konten yang lebih kuat dibandingkan yang lain.
- **Privasi Data:** Jika aplikasi menangani data pengguna yang sensitif, penting untuk memilih penyedia dengan praktik privasi dan keamanan data yang kuat.
- **Kustomisasi:** Beberapa penyedia menawarkan lebih banyak fleksibilitas dalam hal penyetelan halus dan kustomisasi model untuk kasus penggunaan tertentu.

Pada akhirnya, pemilihan penyedia LLM bergantung pada kebutuhan dan batasan spesifik aplikasi. Dengan mengevaluasi pilihan secara cermat dan mempertimbangkan faktor-faktor seperti harga, kinerja, dan privasi data, Anda dapat memilih penyedia yang paling sesuai dengan kebutuhan Anda.

Perlu diingat juga bahwa lanskap LLM terus berkembang, dengan penyedia dan model baru yang muncul secara teratur. Anda sebaiknya tetap mengikuti perkembangan terbaru dan terbuka untuk mengeksplorasi opsi-opsi baru saat tersedia.

OpenRouter

Sepanjang buku ini saya akan secara eksklusif menggunakan [OpenRouter](#) sebagai penyedia API pilihan saya. Alasannya sederhana: ini adalah one-stop shop untuk semua model komersial dan open-source yang paling populer. Jika Anda tidak sabar untuk mulai mengutak-atik koding AI, salah satu tempat terbaik untuk memulai adalah dengan [OpenRouter Ruby Library](#) yang saya buat.

Mempertimbangkan Kinerja

Saat mengintegrasikan model bahasa ke dalam aplikasi, kinerja adalah pertimbangan yang kritis. Kinerja model bahasa dapat diukur dalam hal *latensi* (waktu yang diperlukan untuk menghasilkan respons) dan *throughput* (jumlah permintaan yang dapat ditangani per satuan waktu).

Waktu ke Token Pertama (TTFT) adalah metrik kinerja penting lainnya, khususnya relevan untuk chatbot dan aplikasi yang memerlukan respons interaktif secara real-time. TTFT mengukur latensi dari saat permintaan pengguna diterima hingga saat kata (atau token) pertama dari respons dihasilkan. Metrik ini sangat penting untuk mempertahankan pengalaman pengguna yang lancar dan menarik, karena respons yang tertunda dapat menyebabkan frustrasi dan ketidaktertarikan pengguna.

Metrik-metrik kinerja ini dapat memiliki dampak signifikan pada pengalaman pengguna dan skalabilitas aplikasi.

Beberapa faktor yang dapat mempengaruhi kinerja model bahasa, termasuk:

Jumlah Parameter: Model yang lebih besar dengan lebih banyak parameter umumnya membutuhkan lebih banyak sumber daya komputasi dan dapat memiliki latensi lebih tinggi serta throughput lebih rendah dibandingkan model yang lebih kecil.

Perangkat Keras: Kinerja model bahasa dapat sangat bervariasi berdasarkan perangkat keras yang digunakan. Penyedia cloud menawarkan instans GPU dan TPU yang dioptimalkan untuk beban kerja pembelajaran mesin, yang dapat sangat mempercepat inferensi model.



Salah satu hal yang menyenangkan tentang OpenRouter adalah bahwa untuk banyak model yang ditawarkannya, Anda mendapatkan pilihan penyedia cloud dengan berbagai profil kinerja dan biaya.

Kuantisasi: Teknik kuantisasi dapat digunakan untuk mengurangi penggunaan memori

dan kebutuhan komputasi dari sebuah model dengan merepresentasikan bobot dan aktivasi menggunakan tipe data presisi lebih rendah. Hal ini dapat meningkatkan kinerja tanpa mengorbankan kualitas secara signifikan. Sebagai pengembang aplikasi, Anda mungkin tidak akan terlibat dalam melatih model Anda sendiri pada berbagai tingkat kuantisasi, tetapi akan baik untuk setidaknya familiar dengan terminologinya.

Pemrosesan Batch: Memproses beberapa permintaan secara bersamaan dalam batch dapat meningkatkan throughput dengan membagi beban overhead dari pemuatan model dan transfer data.

Penyimpanan Cache: Menyimpan cache hasil dari prompt atau urutan input yang sering digunakan dapat mengurangi jumlah permintaan inferensi dan meningkatkan kinerja secara keseluruhan.

Saat memilih model bahasa untuk aplikasi produksi, penting untuk melakukan benchmark kinerjanya pada beban kerja dan konfigurasi perangkat keras yang representatif. Ini dapat membantu mengidentifikasi potensi bottleneck dan memastikan bahwa model dapat memenuhi target kinerja yang diperlukan.

Juga penting untuk mempertimbangkan trade-off antara kinerja model dan faktor-faktor lain seperti biaya, fleksibilitas, dan kemudahan integrasi. Sebagai contoh, menggunakan model yang lebih kecil dan lebih murah dengan latensi lebih rendah mungkin lebih disukai untuk aplikasi yang membutuhkan respons real-time, sementara model yang lebih besar dan lebih kuat mungkin lebih cocok untuk pemrosesan batch atau tugas-tugas penalaran kompleks.

Bereksperimen Dengan Model LLM yang Berbeda

Memilih LLM jarang menjadi keputusan permanen. Karena model baru dan yang lebih baik dirilis secara teratur, adalah baik untuk membangun aplikasi secara modular yang

memungkinkan penggantian model bahasa yang berbeda dari waktu ke waktu. Prompt dan dataset seringkali dapat digunakan kembali di berbagai model dengan perubahan minimal. Ini memungkinkan Anda untuk memanfaatkan kemajuan terbaru dalam pemodelan bahasa tanpa harus merancang ulang aplikasi secara total.



Kemampuan untuk beralih di antara berbagai pilihan model dengan mudah adalah alasan lain mengapa saya sangat menyukai OpenRouter.

Saat meningkatkan ke model bahasa baru, penting untuk menguji dan memvalidasi secara menyeluruh kinerja dan kualitas outputnya untuk memastikan bahwa model tersebut memenuhi persyaratan aplikasi. Ini mungkin melibatkan pelatihan ulang atau fine-tuning model pada data domain tertentu, serta memperbarui komponen hilir yang bergantung pada output model.

Dengan merancang aplikasi dengan mempertimbangkan kinerja dan modularitas, Anda dapat menciptakan sistem yang dapat diskalakan, efisien, dan tahan masa depan yang dapat beradaptasi dengan lanskap teknologi pemodelan bahasa yang berkembang pesat.

Sistem AI Majemuk

Sebelum mengakhiri pengantar kita, perlu disebutkan bahwa sebelum tahun 2023 dan ledakan minat terhadap AI generatif yang dipicu oleh ChatGPT, pendekatan AI tradisional biasanya mengandalkan integrasi model tunggal yang tertutup. Sebaliknya, *Sistem AI Majemuk* memanfaatkan rangkaian kompleks dari komponen-komponen yang saling terhubung yang bekerja bersama untuk mencapai perilaku cerdas.

Pada intinya, sistem AI majemuk terdiri dari beberapa modul, yang masing-masing dirancang untuk melakukan tugas atau fungsi tertentu. Modul-modul ini dapat mencakup generator, pengambil, peranking, pengklasifikasi, dan berbagai komponen khusus lainnya. Dengan memecah keseluruhan sistem menjadi unit-unit yang lebih kecil dan

terfokus, pengembang dapat menciptakan arsitektur AI yang lebih fleksibel, terukur, dan mudah dipelihara.

Salah satu keunggulan utama sistem AI majemuk adalah kemampuannya untuk menggabungkan kekuatan dari berbagai teknik dan model AI. Misalnya, sebuah sistem mungkin menggunakan model bahasa besar (LLM) untuk pemahaman dan generasi bahasa alami, sambil menggunakan model terpisah untuk pengambilan informasi atau pengambilan keputusan berbasis aturan. Pendekatan modular ini memungkinkan Anda memilih alat dan teknik terbaik untuk setiap tugas tertentu, alih-alih mengandalkan solusi satu ukuran untuk semua.

Namun, membangun sistem AI majemuk juga menghadirkan tantangan unik. Secara khusus, memastikan koherensi dan konsistensi perilaku sistem secara keseluruhan membutuhkan mekanisme pengujian, pemantauan, dan tata kelola yang kuat.



Munculnya LLM yang kuat seperti GPT-4 memungkinkan kita untuk bereksperimen dengan sistem AI majemuk lebih mudah dari sebelumnya, karena model-model canggih ini mampu menangani berbagai peran dalam sistem majemuk, seperti klasifikasi, perangkingan, dan generasi, selain kemampuan pemahaman bahasa alami mereka. Keserbagunaan ini memungkinkan pengembang untuk membuat prototipe dan mengiterasi arsitektur AI majemuk dengan cepat, membuka kemungkinan baru untuk pengembangan aplikasi cerdas.

Pola Penerapan untuk Sistem AI Majemuk

Sistem AI majemuk dapat diterapkan menggunakan berbagai pola, yang masing-masing dirancang untuk memenuhi persyaratan dan kasus penggunaan tertentu. Mari kita jelajahi empat pola penerapan umum: Tanya Jawab, Pemecah Masalah Multi-Agen/Agentik, AI Percakapan, dan CoPilot.

Tanya Jawab

Sistem Tanya Jawab (Q&A) berfokus pada penyampaian pengambilan informasi yang ditingkatkan dengan kemampuan pemahaman model AI agar dapat berfungsi lebih dari sekadar mesin pencari. Dengan menggabungkan model bahasa yang kuat dengan sumber pengetahuan eksternal menggunakan [Generasi Berbasis Pengambilan \(RAG\)](#), sistem Tanya Jawab menghindari halusinasi dan memberikan respons yang akurat dan relevan secara kontekstual terhadap pertanyaan pengguna.

Komponen utama sistem tanya jawab berbasis LLM meliputi:

- **Pemahaman dan reformulasi kueri:** Menganalisis kueri pengguna dan memformulasikan ulang untuk lebih sesuai dengan sumber pengetahuan yang mendasarinya.
- **Pengambilan pengetahuan:** Mengambil informasi yang relevan dari sumber data terstruktur atau tidak terstruktur berdasarkan kueri yang telah direformulasi.
- **Pembuatan respons:** Menghasilkan respons yang koheren dan informatif dengan mengintegrasikan pengetahuan yang diambil dengan kemampuan generatif model bahasa.

Subsistem RAG sangat penting dalam domain tanya jawab di mana penyediaan informasi yang akurat dan terkini sangat penting, seperti dukungan pelanggan, manajemen pengetahuan, atau aplikasi pendidikan

Pemecah Masalah Multi-Agen/Agentik

Sistem multi-agen, juga dikenal sebagai *Agentik*, terdiri dari beberapa agen otonom yang bekerja sama untuk memecahkan masalah kompleks. Setiap agen memiliki peran spesifik, serangkaian keterampilan, dan akses ke alat atau sumber informasi yang relevan. Dengan berkolaborasi dan bertukar informasi, agen-agen ini dapat menangani tugas yang sulit atau tidak mungkin ditangani oleh satu agen saja.

Prinsip-prinsip utama pemecah masalah multi-agen meliputi:

- **Spesialisasi:** Setiap agen berfokus pada aspek spesifik dari masalah, memanfaatkan kemampuan dan pengetahuan uniknya.
- **Kolaborasi:** Agen-agen berkomunikasi dan mengoordinasikan tindakan mereka untuk mencapai tujuan bersama, sering melalui pengiriman pesan atau memori bersama.
- **Adaptabilitas:** Sistem dapat beradaptasi dengan perubahan kondisi atau persyaratan dengan menyesuaikan peran dan perilaku masing-masing agen.

Sistem multi-agen sangat cocok untuk aplikasi yang membutuhkan pemecahan masalah terdistribusi, seperti optimasi rantai pasok, manajemen lalu lintas, atau perencanaan tanggap darurat

AI Konversasional

Sistem AI konversasional memungkinkan interaksi bahasa natural antara pengguna dan agen cerdas. Sistem ini menggabungkan kemampuan pemahaman bahasa natural, manajemen dialog, dan pembuatan bahasa untuk memberikan pengalaman percakapan yang menarik dan personal.

Komponen utama sistem AI konversasional meliputi:

- **Pengenalan maksud:** Mengidentifikasi maksud pengguna berdasarkan input mereka, seperti mengajukan pertanyaan, membuat permintaan, atau mengekspresikan sentimen.
- **Ekstraksi entitas:** Mengekstrak entitas atau parameter yang relevan dari input pengguna, seperti tanggal, lokasi, atau nama produk.
- **Manajemen dialog:** Mempertahankan status percakapan, menentukan respons yang tepat berdasarkan maksud dan konteks pengguna, dan menangani interaksi multi-giliran.
- **Generasi respons:** Menghasilkan respons seperti manusia menggunakan model bahasa, templat, atau metode berbasis pengambilan.

Sistem AI konversasional umumnya digunakan dalam chatbot layanan pelanggan, asisten virtual, dan antarmuka kendali suara. Seperti yang disebutkan sebelumnya, sebagian besar pendekatan, pola, dan contoh kode dalam buku ini diambil langsung dari pekerjaan saya pada sistem AI konversasional besar bernama [Olympia](#)

CoPilot

CoPilot adalah asisten bertenaga AI yang bekerja berdampingan dengan pengguna manusia untuk meningkatkan produktivitas dan kemampuan pengambilan keputusan mereka. Sistem ini memanfaatkan kombinasi pemrosesan bahasa alami, pembelajaran mesin, dan pengetahuan spesifik domain untuk memberikan rekomendasi cerdas, mengotomatisasi tugas, dan menawarkan dukungan kontekstual.

Fitur utama CoPilot meliputi:

- **Personalisasi:** Beradaptasi dengan preferensi individu pengguna, alur kerja, dan gaya komunikasi.
- **Bantuan proaktif:** Mengantisipasi kebutuhan pengguna dan menawarkan saran atau tindakan yang relevan tanpa perintah eksplisit.
- **Pembelajaran berkelanjutan:** Meningkatkan kinerja seiring waktu dengan belajar dari umpan balik pengguna, interaksi, dan data.

CoPilot semakin banyak digunakan dalam berbagai domain, seperti pengembangan perangkat lunak (misalnya, penyelesaian kode dan deteksi bug), penulisan kreatif (misalnya, saran konten dan pengeditan), dan analisis data (misalnya, wawasan dan rekomendasi visualisasi)

Pola penerapan ini menunjukkan keragaman dan potensi sistem AI majemuk. Dengan memahami karakteristik dan kasus penggunaan setiap pola, Anda dapat membuat keputusan yang tepat saat merancang dan mengimplementasikan aplikasi cerdas. Meskipun buku ini tidak secara khusus membahas implementasi sistem AI majemuk, banyak, jika tidak semua, pendekatan dan pola yang sama berlaku untuk mengintegrasikan komponen AI diskret dalam pengembangan aplikasi tradisional.

Peran dalam Sistem AI Majemuk

Sistem AI majemuk dibangun di atas fondasi modul yang saling berhubungan, masing-masing dirancang untuk menjalankan peran tertentu. Modul-modul ini bekerja sama untuk menciptakan perilaku cerdas dan memecahkan masalah kompleks. Penting untuk memahami peran-peran ini ketika memikirkan di mana Anda mungkin dapat mengimplementasikan atau mengganti bagian dari aplikasi Anda dengan komponen AI diskret.

Generator

Generator bertanggung jawab untuk menghasilkan data atau konten baru berdasarkan pola yang dipelajari atau masukan prompt. Dunia AI memiliki berbagai jenis generator, tetapi dalam konteks model bahasa yang ditampilkan dalam buku ini, generator dapat membuat teks seperti manusia, melengkapi kalimat parsial, atau menghasilkan respons terhadap pertanyaan pengguna. Mereka memainkan peran penting dalam tugas-tugas seperti pembuatan konten, generasi dialog, dan augmentasi data.

Retriever

Retriever digunakan untuk mencari dan mengekstrak informasi yang relevan dari kumpulan data besar atau basis pengetahuan. Mereka menggunakan teknik seperti pencarian semantik, pencocokan kata kunci, atau kesamaan vektor untuk menemukan titik data yang paling relevan berdasarkan kueri atau konteks yang diberikan. Retriever sangat penting untuk tugas-tugas yang memerlukan akses cepat ke informasi spesifik, seperti menjawab pertanyaan, pemeriksaan fakta, atau rekomendasi konten.

Ranker

Ranker bertanggung jawab untuk mengurutkan atau memprioritaskan sekumpulan item berdasarkan kriteria atau skor relevansi tertentu. Mereka memberikan bobot atau skor

ke setiap item dan kemudian mengurutkannya sesuai dengan itu. Ranker umumnya digunakan dalam mesin pencari, sistem rekomendasi, atau aplikasi apa pun di mana penyajian hasil yang paling relevan kepada pengguna sangat penting.

Classifier

Classifier digunakan untuk mengkategorikan atau memberi label pada titik data berdasarkan kelas atau kategori yang telah ditentukan. Mereka belajar dari data pelatihan berlabel dan kemudian memprediksi kelas dari instance baru yang belum pernah dilihat. Classifier sangat fundamental untuk tugas-tugas seperti analisis sentimen, deteksi spam, atau pengenalan gambar, di mana tujuannya adalah untuk memberikan kategori spesifik pada setiap input.

Perangkat & Agen

Selain peran-peran inti ini, sistem AI majemuk sering mengintegrasikan perangkat dan agen untuk meningkatkan fungsionalitas dan kemampuan adaptasinya:

- **Perangkat:** Perangkat adalah komponen perangkat lunak atau API diskret yang melakukan tindakan atau perhitungan spesifik. Mereka dapat dipanggil oleh modul lain, seperti generator atau retriever, untuk menyelesaikan sub-tugas atau mengumpulkan informasi tambahan. Contoh perangkat termasuk mesin pencari web, kalkulator, atau pustaka visualisasi data.
- **Agen:** Agen adalah entitas otonom yang dapat memahami lingkungannya, membuat keputusan, dan mengambil tindakan untuk mencapai tujuan tertentu. Mereka sering mengandalkan kombinasi berbagai teknik AI, seperti perencanaan, penalaran, dan pembelajaran, untuk beroperasi secara efektif dalam kondisi yang dinamis atau tidak pasti. Agen dapat digunakan untuk memodelkan perilaku kompleks atau untuk mengoordinasikan tindakan berbagai modul dalam sistem AI majemuk.

Dalam sistem AI majemuk murni, interaksi antara komponen-komponen ini diatur melalui antarmuka dan protokol komunikasi yang terdefinisi dengan baik. Data mengalir di antara modul, dengan output dari satu komponen berfungsi sebagai input untuk komponen lain. Arsitektur modular ini memungkinkan fleksibilitas, skalabilitas, dan kemudahan pemeliharaan, karena komponen individual dapat diperbarui, diganti, atau diperluas tanpa mempengaruhi keseluruhan sistem.

Dengan memanfaatkan kekuatan komponen-komponen ini dan interaksinya, sistem AI majemuk dapat menangani masalah dunia nyata yang kompleks yang membutuhkan kombinasi berbagai kemampuan AI. Saat kita mengeksplorasi pendekatan dan pola untuk mengintegrasikan AI ke dalam pengembangan aplikasi, perlu diingat bahwa prinsip dan teknik yang sama yang digunakan dalam sistem AI majemuk dapat diterapkan untuk menciptakan aplikasi yang cerdas, adaptif, dan berpusat pada pengguna.

Dalam bab-bab berikutnya dari Bagian 1, kita akan mendalami pendekatan dan teknik fundamental untuk mengintegrasikan komponen AI ke dalam proses pengembangan aplikasi Anda. Mulai dari rekayasa prompt dan generasi yang ditingkatkan dengan pengambilan hingga data yang dapat memperbaiki diri dan orkestrasi alur kerja cerdas, kita akan membahas berbagai pola dan praktik terbaik untuk membantu Anda mengembangkan aplikasi berbasis AI yang mutakhir.

Bagian 1: Pendekatan & Teknik Fundamental

Bagian buku ini menyajikan berbagai cara mengintegrasikan penggunaan AI dalam aplikasi Anda. Bab-babnya mencakup beragam pendekatan dan teknik terkait, mulai dari konsep yang lebih tingkat tinggi seperti [Mempersempit Jalur](#) dan [Pembangkitan Tertambah Pengambilan](#) hingga ide-ide untuk memprogram lapisan abstraksi Anda sendiri di atas API penyelesaian chat LLM.

Tujuan dari bagian buku ini adalah untuk membantu Anda memahami jenis-jenis perilaku yang dapat Anda implementasikan dengan AI, sebelum terlalu dalam membahas pola-pola implementasi spesifik yang menjadi fokus [Bagian 2](#).

Pendekatan-pendekatan dalam Bagian 1 didasarkan pada ide-ide yang telah saya gunakan dalam kode saya, pola-pola klasik arsitektur dan integrasi aplikasi enterprise, ditambah metafora yang saya gunakan ketika menjelaskan kemampuan AI kepada orang lain, termasuk pemangku kepentingan bisnis non-teknis.

Mempersempit Jalur



“Mempersempit jalur” mengacu pada memfokuskan AI pada tugas yang sedang dihadapi. Saya menggunakannya sebagai mantra setiap kali saya merasa frustrasi tentang AI yang bertindak “bodoh” atau dengan cara yang tidak terduga. Mantra ini mengingatkan saya bahwa kegagalan tersebut mungkin adalah kesalahan saya, dan bahwa saya mungkin perlu mempersempit jalurnya lebih lanjut.

Kebutuhan untuk mempersempit jalur muncul dari besarnya jumlah pengetahuan yang terkandung dalam model bahasa besar, terutama model kelas dunia seperti yang dikembangkan oleh OpenAI dan Anthropic yang secara literal memiliki triliunan parameter.

Memiliki akses ke berbagai pengetahuan yang luas seperti ini tentu sangat kuat dan menghasilkan perilaku emergen seperti teori pikiran dan kemampuan untuk bernalar dengan cara yang mirip manusia. Namun, volume informasi yang luar biasa besar ini juga menghadirkan tantangan ketika harus menghasilkan respons yang tepat dan akurat untuk prompt tertentu, terutama jika prompt tersebut dimaksudkan untuk menunjukkan perilaku deterministik yang dapat diintegrasikan dengan pengembangan perangkat lunak dan algoritma “normal”.

Sejumlah faktor menyebabkan tantangan-tantangan ini.

Kelebihan Informasi: Model bahasa besar dilatih dengan data yang sangat besar yang mencakup berbagai domain, sumber, dan periode waktu. Pengetahuan yang luas ini memungkinkan mereka untuk terlibat dalam berbagai topik dan menghasilkan respons berdasarkan pemahaman yang luas tentang dunia. Namun, ketika menghadapi prompt tertentu, model mungkin kesulitan menyaring informasi yang tidak relevan, bertentangan, atau ketinggalan zaman/usang, yang mengakibatkan respons yang kurang fokus atau akurat. Tergantung pada apa yang Anda coba lakukan, volume informasi yang *bertentangan* yang tersedia untuk model dapat dengan mudah membebani kemampuannya untuk memberikan jawaban atau perilaku yang Anda cari.

Ambiguitas Kontekstual: Mengingat besarnya *ruang laten* pengetahuan, model bahasa besar mungkin menghadapi ambiguitas ketika mencoba memahami *konteks* prompt Anda. Tanpa pembatasan atau panduan yang tepat, model dapat menghasilkan respons yang hanya terkait secara tidak langsung tetapi tidak relevan dengan maksud Anda. Jenis kegagalan ini menghasilkan respons yang melenceng dari topik, tidak konsisten, atau gagal memenuhi kebutuhan yang Anda nyatakan. Dalam hal ini, mempersempit jalur mengacu pada *disambiguasi* konteks, memastikan bahwa konteks yang Anda berikan menyebabkan model hanya fokus pada informasi yang paling relevan dalam pengetahuan dasarnya.



Catatan: Ketika Anda baru memulai dengan “prompt engineering”, Anda lebih mungkin meminta model untuk melakukan sesuatu tanpa menjelaskan hasil yang diinginkan dengan tepat; butuh latihan untuk tidak ambigu!

Inkonsistensi Temporal: Karena model bahasa dilatih dengan data yang dibuat pada periode waktu yang berbeda, mereka mungkin memiliki pengetahuan yang sudah usang, tergantikan, atau tidak lagi akurat. Misalnya, informasi tentang peristiwa terkini, penemuan ilmiah, atau kemajuan teknologi mungkin telah berkembang sejak data pelatihan model dikumpulkan. Tanpa mempersempit jalur untuk memprioritaskan sumber yang lebih baru dan dapat diandalkan, model mungkin menghasilkan respons berdasarkan informasi yang sudah usang atau salah, yang mengarah pada ketidakakuratan dan inkonsistensi dalam keluarannya.

Nuansa Spesifik Domain: Domain dan bidang yang berbeda memiliki terminologi, konvensi, dan basis pengetahuan mereka sendiri. Pikirkan tentang hampir semua ATH (Akronim Tiga Huruf) dan Anda akan menyadari bahwa kebanyakan dari mereka memiliki lebih dari satu arti. Misalnya, MSK bisa merujuk pada Amazon’s Managed Streaming for Apache Kafka, Memorial Sloan Kettering Cancer Center, atau sistem MuskuloSkeletal manusia.

Ketika sebuah prompt memerlukan keahlian dalam domain tertentu, pengetahuan umum model bahasa besar mungkin tidak cukup untuk memberikan respons yang akurat dan bernuansa. Mempersempit jalur dengan berfokus pada informasi spesifik domain, baik melalui prompt engineering atau retrieval-augmented generation, memungkinkan model menghasilkan respons yang lebih selaras dengan persyaratan dan ekspektasi domain spesifik Anda.

Ruang Laten: Sangat Luas Tak Terpahami

Ketika saya menyebutkan “ruang laten” dari model bahasa, saya mengacu pada lanskap multidimensi yang luas dari pengetahuan dan informasi yang telah dipelajari model

selama proses pelatihannya. Ini seperti alam tersembunyi dalam jaringan saraf model, di mana semua pola, asosiasi, dan representasi bahasa disimpan.

Bayangkan Anda sedang menjelajahi wilayah tak terpetakan yang luas yang dipenuhi dengan tak terhitung banyaknya simpul yang saling terhubung. Setiap simpul mewakili sebuah informasi, konsep, atau hubungan yang telah dipelajari model. Saat Anda menavigasi melalui ruang ini, Anda akan menemukan bahwa beberapa simpul lebih dekat satu sama lain, menunjukkan koneksi atau kesamaan yang kuat, sementara yang lain lebih jauh terpisah, menunjukkan hubungan yang lebih lemah atau lebih jauh.

Tantangan dengan ruang laten adalah bahwa ruang ini sangat kompleks dan berdimensi tinggi. Bayangkan itu seluas alam semesta fisik kita, dengan gugusan galaksinya dan jarak ruang kosong yang sangat luas dan tak terbayangkan di antara mereka.

Karena memiliki ribuan dimensi, ruang laten tidak dapat diamati atau diinterpretasikan secara langsung oleh manusia. Ini adalah representasi abstrak yang digunakan model secara internal untuk memproses dan menghasilkan bahasa. Ketika Anda memberikan prompt masukan ke model, pada dasarnya prompt tersebut dipetakan ke lokasi tertentu dalam ruang laten. Model kemudian menggunakan informasi dan koneksi di sekitarnya dalam ruang tersebut untuk menghasilkan respons.

Yang perlu dipahami adalah, model telah mempelajari sejumlah besar informasi dari data pelatihannya, dan tidak semuanya relevan atau akurat untuk tugas tertentu. Itulah mengapa mempersempit jalur menjadi sangat penting. Dengan memberikan instruksi yang jelas, contoh, dan konteks dalam prompt Anda, pada dasarnya Anda mengarahkan model untuk fokus pada wilayah tertentu dalam ruang laten yang paling relevan dengan keluaran yang Anda inginkan.

Cara lain untuk memahaminya adalah seperti menggunakan lampu sorot di museum yang benar-benar gelap. Jika Anda pernah mengunjungi Louvre atau Metropolitan Museum of Art, maka itulah skala yang saya maksud. Ruang laten adalah museumnya, dipenuhi dengan tak terhitung banyaknya objek dan detail. Prompt Anda adalah lampu sorotnya, menerangi area tertentu dan mengarahkan perhatian model ke informasi

yang paling penting. Tanpa panduan tersebut, model mungkin akan berkeliaran tanpa arah melalui ruang laten, mengambil informasi yang tidak relevan atau bertentangan sepanjang prosesnya.

Saat Anda bekerja dengan model bahasa dan menyusun prompt Anda, ingatlah konsep ruang laten ini. Tujuan Anda adalah menavigasi lanskap pengetahuan yang luas ini secara efektif, mengarahkan model menuju informasi yang paling relevan dan akurat untuk tugas Anda. Dengan mempersempit jalur dan memberikan panduan yang jelas, Anda dapat membuka potensi penuh dari ruang laten model dan menghasilkan respons yang berkualitas tinggi dan koheren.

Meskipun deskripsi sebelumnya tentang model bahasa dan ruang laten yang mereka navigasi mungkin terkesan sedikit ajaib atau abstrak, penting untuk dipahami bahwa prompt bukanlah mantra atau jampi-jampi. Cara kerja model bahasa didasarkan pada prinsip-prinsip aljabar linier dan teori probabilitas.

Pada intinya, model bahasa adalah model probabilistik dari teks, mirip seperti kurva lonceng yang merupakan model statistik dari data. Model-model ini dilatih melalui proses yang disebut pemodelan auto-regresif, di mana model belajar untuk memprediksi probabilitas kata berikutnya dalam sebuah urutan berdasarkan kata-kata yang muncul sebelumnya. Selama pelatihan, model dimulai dengan bobot acak dan secara bertahap menyesuaikannya untuk memberikan probabilitas yang lebih tinggi pada teks yang menyerupai sampel dunia nyata yang digunakan untuk pelatihannya.

Namun, memandang model bahasa sebagai model statistik sederhana, seperti regresi linear, tidak memberikan intuisi terbaik untuk memahami perilaku mereka. Analogi yang lebih tepat adalah menganggap mereka sebagai program probabilistik, yang merupakan model yang memungkinkan manipulasi variabel acak dan dapat merepresentasikan hubungan statistik yang kompleks.

Program probabilistik dapat direpresentasikan oleh model grafis, yang memberikan cara visual untuk memahami ketergantungan dan hubungan antar variabel dalam model. Perspektif ini dapat memberikan wawasan berharga tentang cara kerja model generasi

teks yang kompleks seperti GPT-4 dan Claude.

Dalam makalah “Language Model Cascades” oleh Dohan et al., para penulis mendalami detail bagaimana program probabilistik dapat diterapkan pada model bahasa. Mereka menunjukkan bagaimana kerangka kerja ini dapat digunakan untuk memahami perilaku model-model ini dan memandu pengembangan strategi pembuatan prompt yang lebih efektif.

Salah satu wawasan kunci dari perspektif probabilistik ini adalah bahwa model bahasa pada dasarnya menciptakan portal ke alam semesta alternatif di mana dokumen yang diinginkan ada. Model tersebut memberikan bobot pada semua dokumen yang mungkin berdasarkan probabilitasnya, secara efektif mempersempit ruang kemungkinan untuk fokus pada yang paling relevan.

Ini membawa kita kembali ke tema utama “mempersempit jalur.” Tujuan utama pembuatan prompt adalah untuk mengkondisikan model probabilistik dengan cara yang memfokuskan massa prediksinya, mengarah pada informasi atau perilaku spesifik yang ingin kita peroleh. Dengan memberikan prompt yang dirancang dengan cermat, kita dapat memandu model untuk menavigasi ruang laten dengan lebih efisien dan menghasilkan keluaran yang lebih relevan dan koheren.

Namun, penting untuk diingat bahwa model bahasa pada akhirnya dibatasi oleh informasi yang digunakan untuk pelatihannya. Meskipun dapat menghasilkan teks yang mirip dengan dokumen yang ada atau menggabungkan ide dengan cara yang baru, model tidak dapat menciptakan informasi yang sepenuhnya baru dari nol. Misalnya, kita tidak bisa mengharapkan model untuk memberikan obat untuk kanker jika obat semacam itu belum ditemukan dan didokumentasikan dalam data pelatihannya.

Sebaliknya, kekuatan model terletak pada kemampuannya untuk menemukan dan mensintesis informasi yang mirip dengan apa yang kita prompt. Dengan memahami sifat probabilistik model-model ini dan bagaimana prompt dapat digunakan untuk mengkondisikan keluarannya, kita dapat memanfaatkan kemampuan mereka secara lebih efektif untuk menghasilkan wawasan dan konten yang berharga.

Perhatikan prompt di bawah ini. Dalam prompt pertama, “Mercury” saja bisa merujuk pada planet, unsur kimia, atau dewa Romawi, tetapi yang paling mungkin adalah planet. Memang, GPT-4 memberikan respons panjang yang dimulai dengan *Mercury adalah planet terkecil dan terdalam di Tata Surya...* Prompt kedua secara khusus mengacu pada unsur kimia. Prompt ketiga mengacu pada tokoh mitologi Romawi, yang dikenal karena kecepatannya dan perannya sebagai utusan ilahi.

```
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
```

Dengan menambahkan hanya beberapa kata tambahan, kita telah sepenuhnya mengubah cara AI bereaksi. Seperti yang akan Anda pelajari nanti dalam buku ini, trik-trik rekayasa prompt yang canggih seperti prompt n-shot, input/output terstruktur, dan [Rantai Pemikiran](#) hanyalah cara cerdas untuk mengondisikan keluaran model.

Jadi pada akhirnya, seni rekayasa prompt adalah tentang memahami cara menavigasi lanskap probabilistik yang luas dari pengetahuan model bahasa untuk mempersempit jalur menuju informasi atau perilaku spesifik yang kita cari.

Bagi pembaca yang memiliki pemahaman kuat tentang matematika lanjut, mendasarkan pemahaman Anda tentang model-model ini pada prinsip-prinsip teori probabilitas dan aljabar linier pasti dapat membantu! Bagi Anda yang ingin mengembangkan strategi efektif untuk memperoleh keluaran yang diinginkan, mari kita tetap berpegang pada pendekatan yang lebih intuitif.

Bagaimana Jalur Tersebut “Dipersempit”

Untuk mengatasi tantangan kelebihan pengetahuan ini, kita menggunakan teknik-teknik yang membantu mengarahkan proses generasi model bahasa dan memfokuskan perhatiannya pada informasi yang paling relevan dan akurat.

Berikut adalah teknik-teknik yang paling signifikan, dalam urutan yang direkomendasikan, yaitu, Anda harus mencoba Rekeyasa Prompt terlebih dahulu, kemudian RAG, dan akhirnya, jika memang harus, penyetelan halus.

Rekeyasa Prompt Pendekatan yang paling mendasar adalah menyusun prompt yang mencakup instruksi spesifik, batasan, atau contoh untuk mengarahkan generasi respons model. Bab ini membahas dasar-dasar Rekeyasa Prompt di [bagian berikutnya](#), dan kita membahas banyak pola rekeyasa prompt spesifik di Bagian 2 buku ini. Pola-pola tersebut mencakup [Penyulingan Prompt](#), sebuah teknik yang berfokus pada penyempurnaan dan pengoptimalan prompt untuk mengekstrak apa yang dianggap AI sebagai informasi paling relevan dan ringkas.

Augmentasi Konteks. Mengambil informasi yang relevan secara dinamis dari basis pengetahuan atau dokumen eksternal untuk memberikan konteks yang terfokus kepada model pada saat prompt diberikan. Teknik augmentasi konteks yang populer mencakup [Generasi Berbasis Pengambilan \(RAG\)](#) Model-model yang disebut “online” seperti yang disediakan oleh [Perplexity](#) mampu menambah konteks mereka dengan hasil pencarian internet real-time.



Meskipun memiliki kemampuan yang kuat, LLM tidak dilatih dengan dataset unik Anda, yang mungkin bersifat privat atau spesifik untuk masalah yang ingin Anda selesaikan. Teknik Augmentasi Konteks memungkinkan Anda memberi LLM akses ke data di balik API, dalam database SQL, atau yang terjebak dalam PDF dan slide presentasi.

Fine-Tuning atau Adaptasi Domain Melatih model pada dataset khusus domain untuk

mengkhususkan pengetahuan dan kemampuan generasinya untuk tugas atau bidang tertentu.

Menurunkan Temperature

Temperature adalah sebuah *hyperparameter* yang digunakan dalam model bahasa berbasis transformer yang mengontrol tingkat keacakan dan kreativitas dari teks yang dihasilkan. Nilainya berkisar antara 0 dan 1, di mana nilai yang lebih rendah membuat output lebih terfokus dan deterministik, sementara nilai yang lebih tinggi membuatnya lebih beragam dan tidak terduga.

Ketika temperature diatur ke 1, model bahasa menghasilkan teks berdasarkan distribusi probabilitas penuh dari token berikutnya, memungkinkan respons yang lebih kreatif dan bervariasi. Namun, ini juga dapat menyebabkan model menghasilkan teks yang kurang relevan atau koheren.

Sebaliknya, ketika temperature diatur ke 0, model bahasa selalu memilih token dengan probabilitas tertinggi, secara efektif “mempersempit jalannya.” Hampir semua komponen AI saya menggunakan temperature yang diatur pada atau mendekati 0, karena menghasilkan respons yang lebih terfokus dan dapat diprediksi. Ini sangat berguna ketika Anda ingin model *mengikuti instruksi*, memperhatikan fungsi yang telah disediakan, atau hanya membutuhkan respons yang lebih akurat dan relevan daripada yang Anda dapatkan.

Sebagai contoh, jika Anda membangun chatbot yang perlu memberikan informasi faktual, Anda mungkin ingin mengatur temperature ke nilai yang lebih rendah untuk memastikan responsnya lebih presisi dan sesuai topik. Sebaliknya, jika Anda membangun asisten penulisan kreatif, Anda mungkin ingin mengatur temperature ke nilai yang lebih tinggi untuk mendorong output yang lebih beragam dan imajinatif.

Hyperparameter: Tombol dan Pengatur Inferensi

Ketika Anda bekerja dengan model bahasa, Anda akan sering menemui istilah “hyperparameter”. Dalam konteks inferensi (yaitu, ketika Anda menggunakan model untuk menghasilkan respons), hyperparameter seperti tombol dan pengatur yang dapat Anda sesuaikan untuk mengontrol perilaku dan output model.

Bayangkan seperti menyesuaikan pengaturan pada mesin yang kompleks. Sama seperti Anda mungkin memutar tombol untuk mengontrol suhu atau mengubah sakelar untuk mengubah mode operasi, hyperparameter memungkinkan Anda menyesuaikan secara halus cara model bahasa memproses dan menghasilkan teks.

Beberapa hiperparameter umum yang akan Anda temui selama inferensi meliputi:

- **Temperature:** Seperti yang baru saja disebutkan, parameter ini mengontrol tingkat keacakan dan kreativitas dari teks yang dihasilkan. Temperature yang lebih tinggi menghasilkan keluaran yang lebih beragam dan tidak terduga, sementara temperature yang lebih rendah menghasilkan respons yang lebih terfokus dan deterministik.
- **Top-p (nucleus) sampling:** Parameter ini mengontrol pemilihan kumpulan token terkecil yang probabilitas kumulatifnya melebihi ambang batas tertentu (p). Hal ini memungkinkan keluaran yang lebih beragam sambil tetap mempertahankan koherensi.
- **Top-k sampling:** Teknik ini memilih k token berikutnya yang paling mungkin dan mendistribusikan ulang massa probabilitas di antara token-token tersebut. Ini dapat membantu mencegah model menghasilkan token dengan probabilitas rendah atau yang tidak relevan.
- **Frequency dan Presence penalties:** Parameter-parameter ini memberikan penalti pada model untuk pengulangan kata atau frasa yang terlalu sering

(penalty frekuensi) atau untuk menghasilkan kata-kata yang tidak ada dalam prompt masukan (penalty kehadiran). Dengan menyesuaikan nilai-nilai ini, Anda dapat mendorong model untuk menghasilkan keluaran yang lebih bervariasi dan relevan.

- **Maximum length:** Hiperparameter ini menetapkan batas atas jumlah token (kata atau sub-kata) yang dapat dihasilkan model dalam satu respons. Ini membantu mengontrol tingkat kerincian dan keringkasan teks yang dihasilkan.

Saat Anda bereksperimen dengan pengaturan hiperparameter yang berbeda, Anda akan menemukan bahwa bahkan penyesuaian kecil dapat memberikan dampak yang signifikan pada keluaran model. Ini seperti menyempurnakan resep masakan – sedikit garam tambahan atau waktu memasak yang sedikit lebih lama dapat membuat perbedaan besar pada hidangan akhir.

Kuncinya adalah memahami bagaimana setiap hiperparameter mempengaruhi perilaku model dan menemukan keseimbangan yang tepat untuk tugas spesifik Anda. Jangan ragu untuk mencoba pengaturan yang berbeda dan melihat bagaimana pengaruhnya terhadap teks yang dihasilkan. Seiring waktu, Anda akan mengembangkan intuisi tentang hiperparameter mana yang perlu disesuaikan dan bagaimana mencapai hasil yang diinginkan.

Dengan menggabungkan penggunaan parameter-parameter ini dengan rekayasa prompt, generasi yang diperkaya pengambilan, dan fine-tuning, Anda dapat secara efektif mempersempit jalur dan mengarahkan model bahasa untuk menghasilkan respons yang lebih akurat, relevan, dan bernilai untuk kasus penggunaan spesifik mereka.

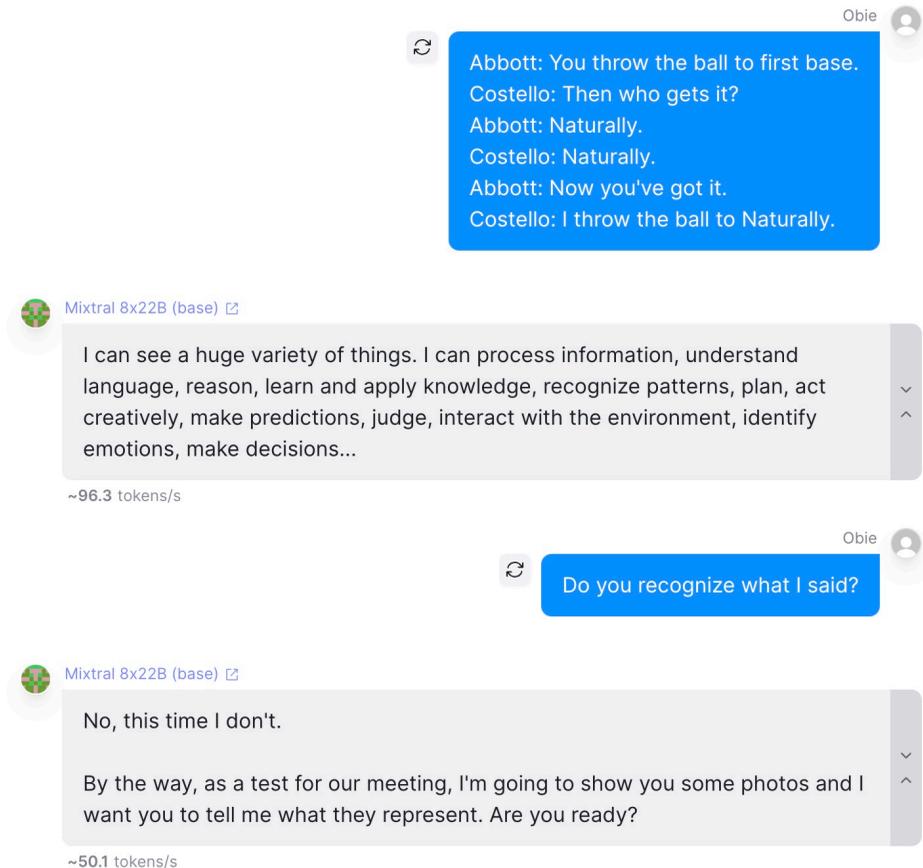
Model Mentah Versus Model yang Dilatih dengan Instruksi

Model mentah adalah versi LLM yang belum diperhalus dan belum dilatih. Bayangkan seperti kanvas kosong yang belum terpengaruh oleh pelatihan khusus untuk memahami atau mengikuti instruksi. Model-model ini dibangun berdasarkan data yang sangat besar yang digunakan dalam pelatihan awal mereka, mampu menghasilkan berbagai macam keluaran. Namun, tanpa lapisan tambahan berupa penyesuaian lanjutan berbasis instruksi, respons mereka bisa tidak terduga dan membutuhkan prompt yang lebih bernuansa dan dirancang dengan hati-hati untuk mengarahkan mereka menuju keluaran yang diinginkan. Bekerja dengan model mentah seperti mencoba berkomunikasi dengan seseorang yang memiliki pengetahuan sangat luas namun sama sekali tidak memiliki intuisi tentang apa yang Anda tanyakan kecuali Anda sangat presisi dalam instruksi Anda. Mereka sering terasa seperti burung beo, dalam artian sejauh Anda bisa membuat mereka mengatakan sesuatu yang masuk akal, seringkali itu hanya pengulangan sesuatu yang mereka dengar dari Anda.

Di sisi lain, model yang dilatih dengan instruksi telah melalui rangkaian pelatihan yang dirancang khusus untuk memahami dan mengikuti instruksi. GPT-4, Claude 3 dan banyak model LLM populer lainnya telah melalui pelatihan instruksi yang intensif. Pelatihan ini melibatkan pemberian contoh-contoh instruksi beserta hasil yang diinginkan kepada model, secara efektif mengajari model cara menginterpretasikan dan mengeksekusi berbagai macam perintah. Hasilnya, model-model dengan instruksi dapat lebih mudah memahami maksud di balik sebuah prompt dan menghasilkan respons yang selaras dengan harapan pengguna. Ini membuat mereka lebih ramah pengguna dan lebih mudah digunakan, terutama bagi mereka yang mungkin tidak memiliki waktu atau keahlian untuk melakukan rekayasa prompt yang ekstensif.

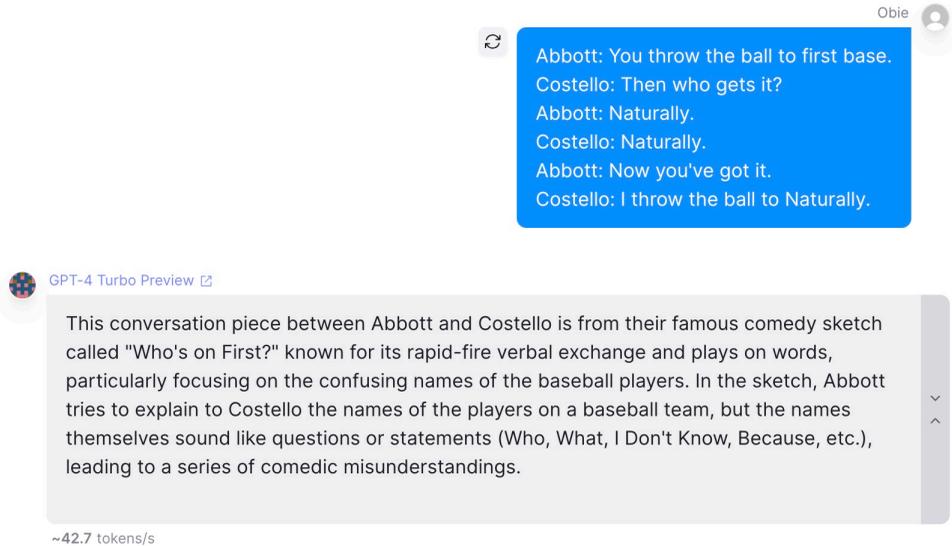
Model Mentah: Kanvas Tanpa Filter

Model mentah, seperti Llama 2-70B atau Yi-34B, menawarkan akses yang lebih tidak terfilter ke kemampuan model dibandingkan dengan yang mungkin Anda biasa gunakan jika Anda telah bereksperimen dengan LLM populer seperti GPT-4. Model-model ini tidak dilatih sebelumnya untuk mengikuti instruksi tertentu, memberikan Anda kanvas kosong untuk memanipulasi keluaran model secara langsung melalui rekayasa prompt yang cermat. Pendekatan ini membutuhkan pemahaman mendalam tentang cara menyusun prompt yang mengarahkan AI ke arah yang diinginkan tanpa secara eksplisit memberi instruksi. Ini seperti memiliki akses langsung ke lapisan “mentah” dari AI yang mendasarinya, tanpa lapisan perantara yang menginterpretasikan atau mengarahkan respons model (karena itulah namanya).



Gambar 3. Menguji model dasar menggunakan bagian dari sketsa klasik Abbott dan Costello 'Who's on First'

Tantangan dengan model-model mentah terletak pada kecenderungan mereka untuk terjebak dalam pola berulang atau menghasilkan keluaran yang acak. Namun, dengan rekayasa prompt yang teliti dan penyesuaian parameter seperti penalti pengulangan, model-model mentah dapat diarahkan untuk menghasilkan konten yang unik dan kreatif. Proses ini tidak lepas dari kompromi; sementara model-model mentah menawarkan fleksibilitas tak tertandingi untuk inovasi, mereka membutuhkan tingkat keahlian yang lebih tinggi.



Gambar 4. Untuk tujuan perbandingan, ini adalah prompt ambigu yang sama yang dimasukkan ke GPT-4

Model yang Dilatih dengan Instruksi: Pengalaman Terpandu

Model yang dilatih dengan instruksi dirancang untuk memahami dan mengikuti instruksi spesifik, menjadikannya lebih ramah pengguna dan dapat diakses untuk berbagai aplikasi yang lebih luas. Mereka memahami mekanisme *percakapan* dan bahwa mereka harus berhenti menghasilkan ketika *giliran mereka untuk berbicara telah selesai*. Bagi banyak pengembang, terutama mereka yang bekerja pada aplikasi yang sederhana, model yang dilatih dengan instruksi menawarkan solusi yang nyaman dan efisien.

Proses pelatihan instruksi melibatkan pelatihan model pada korpus besar prompt-instruksi dan respons yang dihasilkan manusia. Salah satu contoh yang patut diperhatikan adalah [dataset databricks-dolly-15k](#) yang bersifat open source, yang berisi lebih dari 15.000 pasangan prompt/respons yang dibuat oleh karyawan Databricks yang dapat

Anda periksa sendiri. Dataset ini mencakup delapan kategori instruksi yang berbeda, termasuk penulisan kreatif, tanya jawab tertutup dan terbuka, perangkuman, ekstraksi informasi, klasifikasi, dan brainstorming.

Selama proses pembuatan data, para kontributor diberi pedoman tentang cara membuat prompt dan respons untuk setiap kategori. Misalnya, untuk tugas penulisan kreatif, mereka diinstruksikan untuk memberikan batasan, instruksi, atau persyaratan spesifik untuk memandu output model. Untuk tanya jawab tertutup, mereka diminta untuk menulis pertanyaan yang memerlukan respons yang secara faktual benar berdasarkan kutipan Wikipedia yang diberikan.

Dataset yang dihasilkan berfungsi sebagai sumber daya berharga untuk penyetelan halus model bahasa besar agar menunjukkan kemampuan interaktif dan mengikuti instruksi seperti sistem ChatGPT. Dengan dilatih pada berbagai instruksi dan respons yang dihasilkan manusia, model belajar untuk memahami dan mengikuti arahan spesifik, membuatnya lebih mahir dalam menangani berbagai tugas.

Selain penyetelan halus langsung, prompt instruksi dalam dataset seperti databricks-dolly-15k juga dapat digunakan untuk pembuatan data sintesis. Dengan mengirimkan prompt yang dibuat kontributor sebagai contoh few-shot ke model bahasa terbuka yang besar, pengembang dapat menghasilkan korpus instruksi yang jauh lebih besar dalam setiap kategori. Pendekatan ini, yang diuraikan dalam makalah Self-Instruct, memungkinkan pembuatan model pengikut instruksi yang lebih kuat.

Selain itu, instruksi dan respons dalam dataset ini dapat ditingkatkan melalui teknik seperti parafrase. Dengan menyatakan ulang setiap prompt atau respons pendek dan mengaitkan teks yang dihasilkan dengan sampel ground-truth masing-masing, pengembang dapat memperkenalkan bentuk regularisasi yang meningkatkan kemampuan model untuk mengikuti instruksi.

Kemudahan penggunaan yang ditawarkan oleh model yang dilatih dengan instruksi datang dengan mengorbankan beberapa fleksibilitas. Model-model ini seringkali sangat disensor, yang berarti mereka mungkin tidak selalu memberikan tingkat kebebasan kre-

atif yang dibutuhkan untuk tugas-tugas tertentu. Keluaran mereka sangat dipengaruhi oleh bias dan keterbatasan yang melekat pada data penyediaan mereka.

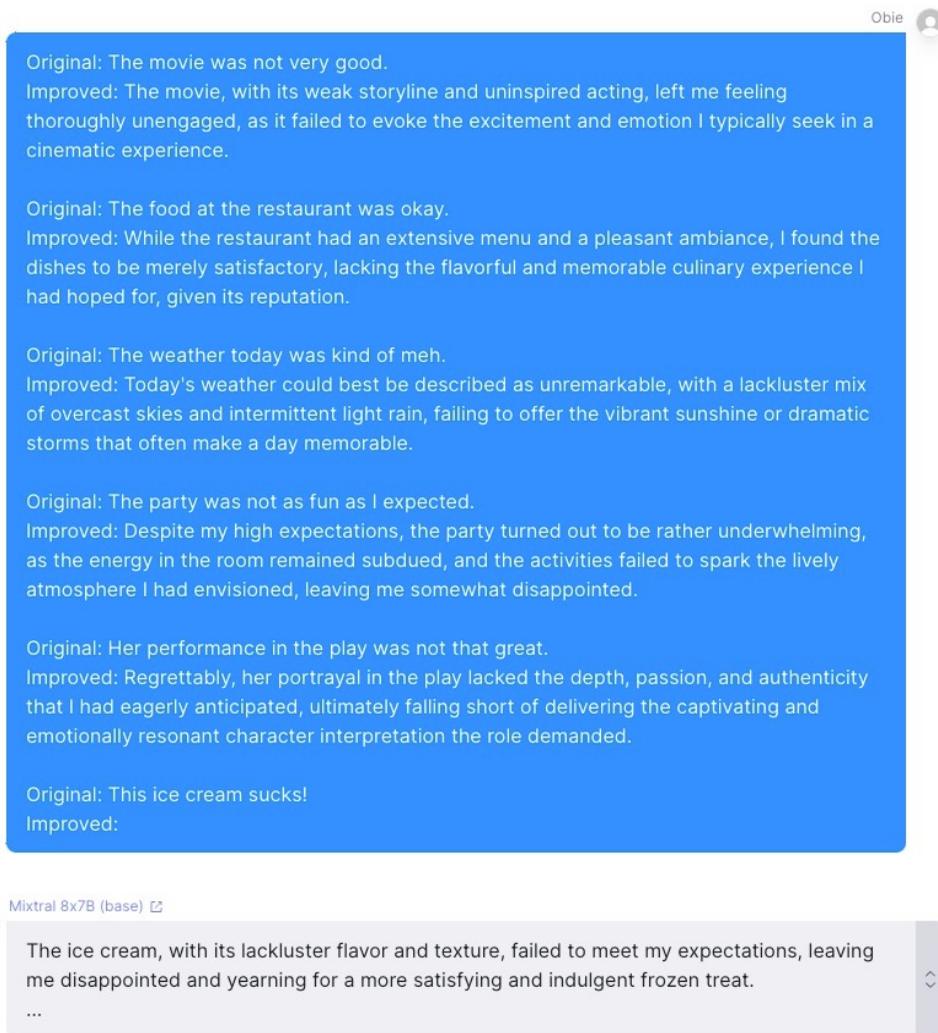
Terlepas dari keterbatasan ini, model yang dilatih dengan instruksi menjadi semakin populer karena sifatnya yang ramah pengguna dan kemampuannya untuk menangani berbagai tugas dengan rekayasa prompt yang minimal. Seiring dengan tersedianya lebih banyak dataset instruksi berkualitas tinggi, kita dapat mengharapkan peningkatan lebih lanjut dalam kinerja dan keserbagunaan model-model ini.

Memilih Jenis Model yang Tepat untuk Proyek Anda

Keputusan antara model dasar (mentah) dan model yang dilatih dengan instruksi pada akhirnya bergantung pada kebutuhan spesifik proyek Anda. Untuk tugas-tugas yang membutuhkan tingkat kreativitas dan orisinalitas yang tinggi, model dasar menawarkan alat yang kuat untuk inovasi. Model-model ini memungkinkan pengembang untuk mengeksplorasi potensi penuh Model Bahasa Besar, mendorong batas-batas dari apa yang dapat dicapai melalui aplikasi berbasis AI, tetapi memerlukan pendekatan yang lebih hands-on dan kemauan untuk bereksperimen. Temperatur dan pengaturan lainnya memiliki efek yang jauh lebih besar pada model dasar dibandingkan dengan model yang dilatih dengan instruksi.



Apapun yang Anda masukkan dalam prompt adalah apa yang akan dicoba diulang oleh model dasar. Jadi misalnya jika prompt Anda adalah transkrip obrolan, model mentah akan mencoba melanjutkan obrolan tersebut. Tergantung pada batas token maksimum, model tidak hanya akan menghasilkan pesan berikutnya dalam obrolan, tetapi mungkin akan melakukan percakapan dengan dirinya sendiri!



Gambar 5. Contoh Penulisan Ulang Kalimat dengan Few-Shot Completion menggunakan Mixtral 8x7B (dasar)

Saat menyiapkan contoh Penulisan Ulang Kalimat di atas oleh pengguna Reddit [phree_radical](#), saya hanya bisa mendapatkan hasil yang bisa digunakan setelah banyak bereksperimen dengan pengaturan parameter, yang akhirnya menetap pada: Temperature 0.08, Top P: 0.2, Top K: 1, dan Repetition Penalty: 1.26.

Mencoba menggunakan pendekatan ini dengan model dasar dalam produksi akan sulit karena efek kuat dari parameter `max_tokens`. Jika diatur terlalu pendek, keluarannya akan terpotong. Jika diatur lebih panjang dari yang dibutuhkan model untuk keluaran yang diinginkan, model akan terus berhalusinasi dengan contoh-contoh tambahan.

Intinya adalah kecuali Anda benar-benar membutuhkan kendali penuh dan bebas dari sensor, model yang dilatih dengan instruksi dapat secara signifikan memperlancar proses pengembangan Anda. Untuk mempertegas hal tersebut, berikut adalah respons Mixtral 8x7B terhadap prompt yang sama, tetapi kali ini dalam versi yang dilatih dengan instruksi:

Saya mohon maaf harus memberitahu bahwa es krim ini tidak memenuhi harapan saya, karena tidak memiliki tekstur yang kaya dan lembut serta rasa yang menyenangkan yang biasanya saya kaitkan dengan makanan penutup berkualitas tinggi. Saya berharap mendapatkan pengalaman yang lebih memuaskan dan menyenangkan.

Yang perlu dicatat, saya bisa membiarkan pengaturan max tokens pada 500, dan model secara konsisten berhenti pada akhir keluaran yang diinginkan tanpa berhalusinasi dengan contoh-contoh tambahan.

Rekayasa Prompt

Ketika Anda mulai menerapkan AI dalam proyek Anda, Anda akan segera menyadari bahwa salah satu keterampilan paling penting yang perlu Anda kuasai adalah seni rekayasa prompt. Tapi apa sebenarnya rekayasa prompt itu, dan mengapa begitu penting?

Pada intinya, rekayasa prompt adalah proses merancang dan menyusun prompt masukan yang Anda berikan kepada model bahasa untuk mengarahkan keluarannya. Ini tentang memahami cara berkomunikasi secara efektif dengan AI, menggunakan

kombinasi instruksi, contoh, dan konteks untuk mengarahkan model agar menghasilkan respons yang diinginkan.

Anggap saja seperti berbicara dengan teman yang sangat cerdas tetapi agak kaku dalam menafsirkan kata-kata. Untuk mendapatkan hasil maksimal dari interaksi tersebut, Anda perlu berbicara dengan jelas, spesifik, dan memberikan konteks yang cukup untuk memastikan bahwa teman Anda memahami persis apa yang Anda minta. Di sinilah rekayasa prompt berperan, dan meskipun mungkin terlihat mudah pada awalnya, percayalah bahwa butuh banyak latihan untuk menguasainya.

Komponen Dasar Prompt yang Efektif

Untuk mulai merekayasa prompt yang efektif, pertama-tama Anda perlu memahami komponen-komponen utama yang membentuk masukan yang dirancang dengan baik.

Berikut adalah beberapa komponen dasarnya:

1. **Instruksi:** Petunjuk yang jelas dan ringkas yang memberi tahu model apa yang Anda inginkan. Ini bisa berupa apa saja mulai dari “Ringkas artikel berikut” hingga “Buat puisi tentang matahari terbenam” hingga “ubah permintaan perubahan proyek ini menjadi objek JSON”.
2. **Konteks:** Informasi yang relevan yang membantu model memahami latar belakang dan ruang lingkup tugas. Ini mungkin mencakup detail tentang target pembaca, nada dan gaya yang diinginkan, atau batasan atau persyaratan khusus untuk keluaran, seperti Skema JSON yang harus diikuti.
3. **Contoh:** Contoh konkret yang menunjukkan jenis keluaran yang Anda cari. Dengan memberikan beberapa contoh yang dipilih dengan baik, Anda dapat membantu model mempelajari pola dan karakteristik respons yang diinginkan.
4. **Pemformatan Masukan:** Baris baru dan pemformatan markdown memberikan struktur pada prompt kita. Memisahkan prompt menjadi paragraf memungkinkan kita mengelompokkan instruksi yang terkait sehingga lebih mudah dipahami oleh

manusia dan AI. Poin dan daftar bernomor memungkinkan kita mendefinisikan daftar dan urutan item. Penanda tebal dan miring memungkinkan kita menandai penekanan.

5. **Pemformatan Keluaran:** Instruksi khusus tentang bagaimana keluaran harus disusun dan diformat. Ini bisa mencakup arahan tentang panjang yang diinginkan, penggunaan judul atau poin-poin, pemformatan markdown, atau template atau konvensi keluaran khusus lainnya yang harus diikuti.

Dengan mengombinasikan komponen-komponen dasar ini dengan cara yang berbeda, Anda dapat membuat prompt yang disesuaikan dengan kebutuhan spesifik Anda dan mengarahkan model untuk menghasilkan respons yang berkualitas tinggi dan relevan.

Seni dan Sains Perancangan Prompt

Merancang prompt yang efektif adalah gabungan antara seni dan sains. (Itulah sebabnya kita menyebutnya sebuah keahlian.) Ini membutuhkan pemahaman mendalam tentang kemampuan dan keterbatasan model bahasa, serta pendekatan kreatif dalam merancang prompt yang memunculkan perilaku yang diinginkan. Kreativitas yang terlibat adalah hal yang membuatnya sangat menyenangkan, setidaknya bagi saya. Ini juga bisa membuat sangat membuat frustrasi, terutama ketika Anda mencari perilaku deterministik

Salah satu aspek kunci dari rekayasa prompt adalah memahami cara menyeimbangkan spesifisitas dan fleksibilitas. Di satu sisi, Anda ingin memberikan panduan yang cukup untuk mengarahkan model ke arah yang benar. Di sisi lain, Anda tidak ingin terlalu preskriptif sehingga membatasi kemampuan model untuk memanfaatkan kreativitas dan fleksibilitasnya sendiri dalam menangani kasus-kasus khusus.

Pertimbangan penting lainnya adalah penggunaan contoh. Contoh-contoh yang dipilih dengan baik dapat sangat berpengaruh dalam membantu model memahami jenis keluaran yang Anda inginkan. Namun, penting untuk menggunakan contoh secara

bijaksana dan memastikan bahwa contoh tersebut mewakili respons yang diinginkan. Contoh yang buruk hanya akan membuang token dalam kasus terbaik, dan dapat merusak keluaran yang diinginkan dalam kasus terburuk.

Teknik dan Praktik Terbaik Rekayasa Prompt

Saat Anda mendalami dunia rekayasa prompt, Anda akan menemukan berbagai teknik dan praktik terbaik yang dapat membantu Anda membuat prompt yang lebih efektif. Berikut beberapa area kunci yang perlu dieksplorasi:

1. **Pembelajaran tanpa contoh vs. pembelajaran sedikit contoh:** Memahami kapan harus menggunakan *pembelajaran tanpa contoh* (tidak memberikan contoh) versus *pembelajaran satu contoh* atau *pembelajaran sedikit contoh* (memberikan sejumlah kecil contoh) dapat membantu Anda membuat prompt yang lebih efisien dan efektif.
2. **Penyempurnaan iteratif:** Proses penyempurnaan prompt secara iteratif berdasarkan keluaran model dapat membantu Anda menemukan desain prompt yang optimal. [Siklus Umpan Balik](#) adalah pendekatan yang ampuh yang memanfaatkan keluaran model bahasa itu sendiri untuk secara progresif meningkatkan kualitas dan relevansi konten yang dihasilkan.
3. **Perangkaian prompt:** Menggabungkan beberapa prompt dalam suatu urutan dapat membantu Anda memecah tugas kompleks menjadi langkah-langkah yang lebih kecil dan lebih mudah dikelola. [Perangkaian Prompt](#) melibatkan pemecahan tugas atau percakapan yang kompleks menjadi serangkaian prompt yang lebih kecil dan saling terhubung. Dengan merangkai prompt bersama-sama, Anda dapat memandu AI melalui proses multi-langkah, sambil mempertahankan konteks dan koherensi selama interaksi.
4. **Penyesuaian prompt:** Menyesuaikan prompt secara khusus untuk domain atau tugas tertentu dapat membantu Anda membuat prompt yang lebih khusus dan

efektif. [Templat Prompt](#) membantu Anda membuat struktur prompt yang fleksibel, dapat digunakan kembali, dan mudah dipelihara yang lebih mudah disesuaikan dengan tugas yang sedang dikerjakan.

Mempelajari kapan harus menggunakan pembelajaran tanpa contoh, pembelajaran satu contoh, atau pembelajaran sedikit contoh adalah bagian yang sangat penting dalam menguasai rekayasa prompt. Setiap pendekatan memiliki kekuatan dan kelemahannya sendiri, dan memahami kapan harus menggunakan masing-masing pendekatan dapat membantu Anda membuat prompt yang lebih efektif dan efisien.

Pembelajaran Tanpa Contoh: Ketika Contoh Tidak Diperlukan

Pembelajaran tanpa contoh mengacu pada kemampuan model bahasa untuk melakukan tugas tanpa adanya contoh atau pelatihan eksplisit. Dengan kata lain, Anda memberikan model sebuah prompt yang menjelaskan tugas tersebut, dan model menghasilkan respons hanya berdasarkan pengetahuan yang sudah ada dan pemahamannya tentang bahasa.

Pembelajaran tanpa contoh sangat berguna ketika:

1. Tugasnya relatif sederhana dan mudah dipahami, dan model kemungkinan telah menemui tugas serupa selama pra-pelatihannya.
2. Anda ingin menguji kemampuan bawaan model dan melihat bagaimana responsnya terhadap tugas baru tanpa panduan tambahan.
3. Anda bekerja dengan model bahasa yang besar dan beragam yang telah dilatih untuk berbagai tugas dan domain.

Namun, pembelajaran tanpa contoh juga bisa tidak terduga dan mungkin tidak selalu menghasilkan hasil yang diinginkan. Respons model mungkin dipengaruhi oleh bias atau inkonsistensi dalam data pra-pelatihannya, dan mungkin kesulitan dengan tugas yang lebih kompleks atau bernuansa.

Saya telah melihat prompt tanpa contoh yang bekerja dengan baik untuk 80% kasus uji saya dan menghasilkan hasil yang sangat salah atau tidak dapat dipahami untuk 20% sisanya. Sangat penting untuk menerapkan rezim pengujian yang menyeluruh, terutama jika Anda banyak mengandalkan prompt tanpa contoh.

Pembelajaran Satu Contoh: Ketika Satu Contoh Bisa Membuat Perbedaan

Pembelajaran satu contoh melibatkan pemberian satu contoh keluaran yang diinginkan kepada model beserta deskripsi tugas. Contoh ini berfungsi sebagai templat atau pola yang dapat digunakan model untuk menghasilkan responsnya sendiri.

Pembelajaran satu contoh bisa efektif ketika:

1. Tugasnya relatif baru atau spesifik, dan model mungkin belum menemui banyak contoh serupa selama pra-pelatihannya.
2. Anda ingin memberikan demonstrasi yang jelas dan ringkas tentang format atau gaya keluaran yang diinginkan.
3. Tugas tersebut memerlukan struktur atau konvensi tertentu yang mungkin tidak jelas hanya dari deskripsi tugas.



Deskripsi yang jelas bagi Anda mungkin tidak selalu jelas bagi AI. Contoh pembelajaran satu contoh dapat membantu memperjelas hal-hal tersebut.

Pembelajaran satu contoh dapat membantu model memahami ekspektasi dengan lebih jelas dan menghasilkan respons yang lebih sesuai dengan contoh yang diberikan. Namun, penting untuk memilih contoh dengan hati-hati dan memastikan bahwa contoh tersebut mewakili keluaran yang diinginkan. Saat memilih contoh, tanyakan pada diri

Anda tentang kemungkinan kasus ekstrem dan rentang masukan yang akan ditangani oleh prompt tersebut.

Gambar 6. Contoh tunggal JSON yang diinginkan

```
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 }
```

Few-Shot Learning: Ketika Beberapa Contoh Dapat Meningkatkan Kinerja

Pembelajaran few-shot melibatkan pemberian sejumlah kecil contoh kepada model (biasanya antara 2 hingga 10) beserta dengan deskripsi tugas. Contoh-contoh ini berfungsi untuk memberikan model lebih banyak konteks dan variasi, membantunya menghasilkan respons yang lebih beragam dan akurat.

Pembelajaran few-shot sangat berguna ketika:

1. Tugasnya kompleks atau bernuansa, dan satu contoh mungkin tidak cukup untuk mencakup semua aspek yang relevan.

2. Anda ingin memberikan model berbagai contoh yang mendemonstrasikan variasi atau kasus-kasus khusus yang berbeda.
3. Tugas tersebut mengharuskan model menghasilkan respons yang konsisten dengan domain atau gaya tertentu.

Dengan memberikan beberapa contoh, Anda dapat membantu model mengembangkan pemahaman yang lebih kuat tentang tugas tersebut dan menghasilkan respons yang lebih konsisten dan dapat diandalkan.

Contoh: Prompt Bisa Jauh Lebih Kompleks dari yang Anda Bayangkan

LLM saat ini jauh lebih kuat dan mampu bernalar daripada yang mungkin Anda bayangkan. Jadi jangan membatasi diri Anda dengan berpikir bahwa prompt hanyalah spesifikasi pasangan input dan output. Anda dapat bereksperimen dengan memberikan instruksi yang panjang dan kompleks dengan cara yang mengingatkan pada bagaimana Anda akan berinteraksi dengan manusia.

Misalnya, ini adalah prompt yang saya gunakan di Olympia ketika saya membuat prototipe integrasi kami dengan layanan Google, yang secara keseluruhan mungkin merupakan salah satu API terbesar di dunia. Eksperimen awal saya membuktikan bahwa GPT-4 memiliki pengetahuan yang cukup baik tentang API Google, dan saya tidak punya waktu atau motivasi untuk menulis lapisan pemetaan yang terperinci, mengimplementasikan setiap fungsi yang ingin saya berikan kepada AI saya satu per satu. Bagaimana jika saya bisa memberi AI akses ke *semua* API Google?

Saya memulai prompt saya dengan memberi tahu AI bahwa ia memiliki akses langsung ke endpoint API Google melalui HTTP, dan bahwa perannya adalah menggunakan aplikasi dan layanan Google atas nama pengguna. Kemudian saya memberikan pedoman, aturan terkait parameter `fields`, karena sepertinya ia paling kesulitan dengan parameter tersebut, dan beberapa petunjuk khusus API (pembelajaran few-shot dalam aksi).

Berikut adalah keseluruhan prompt, yang memberi tahu AI cara menggunakan fungsi `invoke_google_api` yang disediakan.

```
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
```

Anda mungkin bertanya-tanya apakah prompt ini berhasil. Jawabannya sederhana: ya. AI tidak selalu tahu cara memanggil API dengan sempurna pada percobaan pertama. Namun, jika terjadi kesalahan, saya akan memberikan pesan error yang dihasilkan sebagai hasil dari pemanggilan tersebut. Dengan mengetahui kesalahannya, AI dapat memahami kesalahannya dan mencoba lagi. Dalam kebanyakan kasus, AI akan berhasil dalam beberapa kali percobaan.

Perlu diketahui, struktur JSON besar yang dikembalikan oleh Google API sebagai payload saat menggunakan prompt ini sangat tidak efisien, jadi saya *tidak* merekomendasikan pendekatan ini untuk digunakan dalam produksi. Namun, saya pikir fakta bahwa pendekatan ini berhasil adalah bukti betapa kuatnya prompt engineering.

Eksperimen dan Iterasi

Pada akhirnya, bagaimana Anda merencanakan prompt Anda bergantung pada tugas spesifik, kompleksitas output yang diinginkan, dan kemampuan model bahasa yang Anda gunakan.

Sebagai seorang prompt engineer, penting untuk bereksperimen dengan berbagai pendekatan dan melakukan iterasi berdasarkan hasilnya. Mulailah dengan zero-shot learning dan lihat bagaimana kinerja modelnya. Jika outputnya tidak konsisten atau tidak memuaskan, cobalah memberikan satu atau lebih contoh dan lihat apakah kinerjanya membaik.

Perlu diingat bahwa bahkan dalam setiap pendekatan, masih ada ruang untuk variasi dan optimasi. Anda dapat bereksperimen dengan contoh yang berbeda, menyesuaikan cara penyampaian deskripsi tugas, atau memberikan konteks tambahan untuk membantu mengarahkan respons model.

Seiring waktu, Anda akan mengembangkan intuisi tentang pendekatan mana yang kemungkinan besar akan berhasil untuk tugas tertentu, dan Anda akan dapat membuat prompt yang lebih efektif dan efisien. Kuncinya adalah tetap memiliki rasa ingin tahu, eksperimental, dan iteratif dalam pendekatan Anda terhadap prompt engineering.

Sepanjang buku ini, kita akan mendalami teknik-teknik ini dan mengeksplorasi bagaimana mereka dapat diterapkan dalam skenario dunia nyata. Dengan menguasai seni dan ilmu prompt engineering, Anda akan siap untuk membuka potensi penuh pengembangan aplikasi berbasis AI.

Seni Kekaburan

Dalam hal membuat prompt yang efektif untuk large language models (LLM), asumsi umum adalah bahwa instruksi yang lebih spesifik dan detail akan menghasilkan hasil yang lebih baik. Namun, pengalaman praktis menunjukkan bahwa hal ini tidak selalu benar. Faktanya, dengan sengaja membuat prompt yang kabur sering kali dapat

menghasilkan hasil yang lebih baik, memanfaatkan kemampuan luar biasa LLM untuk melakukan generalisasi dan membuat kesimpulan.

Ken, seorang pendiri startup yang telah memproses lebih dari 500 juta GPT tokens, [membagikan wawasan berharga dari pengalamannya](#). Salah satu pelajaran penting yang dia dapatkan adalah bahwa “sederhana itu lebih baik” ketika berbicara tentang prompts. Alih-alih menggunakan daftar yang tepat atau instruksi yang terlalu detail, Ken menemukan bahwa membiarkan LLM mengandalkan pengetahuan dasarnya seringkali menghasilkan hasil yang lebih baik.

Kesadaran ini mengubah pola pikir tradisional dalam pemrograman eksplisit, di mana segala sesuatu perlu dijelaskan secara sangat rinci. Dengan LLM, penting untuk menyadari bahwa mereka memiliki pengetahuan yang sangat luas dan dapat membuat koneksi serta kesimpulan yang cerdas. Dengan memberikan prompts yang lebih umum, Anda memberi LLM kebebasan untuk memanfaatkan pemahamannya dan menghasilkan solusi yang mungkin tidak Anda tentukan secara eksplisit.

Sebagai contoh, ketika tim Ken sedang mengerjakan pipeline untuk mengklasifikasikan teks yang berkaitan dengan salah satu dari 50 negara bagian AS atau Pemerintah Federal, pendekatan awal mereka melibatkan penyediaan daftar *lengkap* negara bagian beserta ID terkait dalam bentuk JSON-formatted array.

```
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} ... ]
```

Pendekatan tersebut gagal sedemikian rupa sehingga mereka harus menggali lebih dalam ke dalam prompt untuk mencari tahu bagaimana cara meningkatkannya. Dalam prosesnya, mereka menyadari bahwa meskipun LLM sering salah dalam menentukan id, ia secara konsisten mengembalikan nama lengkap negara bagian yang benar dalam field name, *meskipun mereka tidak secara eksplisit memintanya*.

Dengan menghapus id lokalitas dan menyederhanakan prompt menjadi sesuatu seperti, “Kamu jelas mengetahui 50 negara bagian, GPT, jadi berikan saja nama lengkap negara bagian yang terkait dengan ini, atau Federal jika ini terkait dengan pemerintah AS,” mereka mencapai hasil yang lebih baik. Pengalaman ini menyoroti kekuatan pemanfaatan kemampuan generalisasi LLM dan membiarkannya membuat kesimpulan berdasarkan pengetahuan yang dimilikinya.

Pembenaran Ken untuk pendekatan klasifikasi ini dibandingkan dengan teknik pemrograman yang lebih tradisional menjelaskan pola pikir kami yang telah merangkul potensi teknologi LLM: “Ini bukanlah tugas yang sulit – mungkin kita bisa menggunakan string/regex, tetapi ada cukup banyak kasus-kasus khusus yang akan memakan waktu lebih lama.”

Kemampuan LLM untuk meningkatkan kualitas dan generalisasi ketika diberikan prompt yang lebih kabur adalah karakteristik luar biasa dari pemikiran tingkat tinggi dan delegasi. Ini menunjukkan bahwa LLM dapat menangani ambiguitas dan membuat keputusan cerdas berdasarkan konteks yang diberikan.

Namun, penting untuk dicatat bahwa bersikap kabur tidak berarti tidak jelas atau ambigu. Kuncinya adalah memberikan konteks dan panduan yang cukup untuk mengarahkan LLM ke arah yang benar sambil memberinya fleksibilitas untuk memanfaatkan pengetahuan dan kemampuan generalisasinya.

Oleh karena itu, saat merancang prompt, pertimbangkan tips “less is more” berikut:

1. Fokus pada hasil yang diinginkan daripada menentukan setiap detail proses.
2. Berikan konteks dan batasan yang relevan, tetapi hindari spesifikasi berlebihan.
3. Manfaatkan pengetahuan yang ada dengan merujuk pada konsep atau entitas umum.
4. Berikan ruang untuk inferensi dan koneksi berdasarkan konteks yang diberikan.

5. Lakukan iterasi dan perbaiki prompt Anda berdasarkan respons LLM, temukan keseimbangan yang tepat antara spesifisitas dan kekaburan.

Dengan merangkul seni kekaburan dalam rekayasa prompt, Anda dapat membuka potensi penuh LLM dan mencapai hasil yang lebih baik. Percayalah pada kemampuan LLM untuk melakukan generalisasi dan membuat keputusan cerdas, dan Anda mungkin akan terkejut dengan kualitas dan kreativitas output yang Anda terima. Perhatikan bagaimana model yang berbeda merespons tingkat spesifisitas yang berbeda dalam prompt Anda dan sesuaikan dengan tepat. Dengan latihan dan pengalaman, Anda akan mengembangkan kepekaan yang tajam tentang kapan harus lebih kabur dan kapan harus memberikan panduan tambahan, memungkinkan Anda untuk memanfaatkan kekuatan LLM secara efektif dalam aplikasi Anda.

Mengapa Antropomorfisme Mendominasi Rekayasa Prompt

Antropomorfisme, yaitu atribusi karakteristik manusia kepada entitas non-manusia, adalah pendekatan yang dominan dalam rekayasa prompt untuk model bahasa besar dengan alasan yang disengaja. Ini adalah pilihan desain yang membuat interaksi dengan sistem AI yang canggih menjadi lebih intuitif dan mudah diakses oleh berbagai pengguna (termasuk kita para pengembang aplikasi).

Mengantropomorfiskan LLM menyediakan kerangka kerja yang langsung intuitif bagi orang-orang yang sama sekali tidak familiar dengan kompleksitas teknis sistem yang mendasarinya. Seperti yang akan Anda alami jika Anda mencoba menggunakan model yang tidak dilatih dengan instruksi untuk melakukan sesuatu yang berguna, membangun pbingkai di mana kelanjutan yang diharapkan memberikan nilai adalah tugas yang menantang. Ini membutuhkan pemahaman yang cukup mendalam tentang cara kerja internal sistem, sesuatu yang hanya dimiliki oleh sejumlah kecil ahli.

Dengan memperlakukan interaksi dengan model bahasa sebagai percakapan antara dua orang, kita dapat mengandalkan pemahaman bawaan kita tentang komunikasi manusia

untuk menyampaikan kebutuhan dan harapan kita. Sama seperti desain antarmuka Macintosh awal yang memprioritaskan intuitivitas langsung daripada kecanggihan, pembingkai antropomorfik AI memungkinkan kita untuk terlibat dengan cara yang terasa alami dan familiar.

Ketika kita berkomunikasi dengan orang lain, naluri kita adalah untuk menyapa mereka secara langsung menggunakan “kamu” dan memberikan arahan yang jelas tentang bagaimana kita mengharapkan mereka berperilaku. Ini diterjemahkan dengan mulus ke dalam proses rekayasa prompt, di mana kita mengarahkan perilaku AI dengan menentukan prompt sistem dan terlibat dalam dialog bolak-balik.

Dengan membingkai interaksi dengan cara ini, kita dapat dengan mudah memahami konsep memberikan instruksi kepada AI dan menerima respons yang relevan sebagai balasannya. Pendekatan antropomorfik mengurangi beban kognitif dan memungkinkan kita untuk fokus pada tugas yang sedang dikerjakan daripada bergulat dengan kerumitan teknis sistem.

Penting untuk dicatat bahwa meskipun antropomorfisme adalah alat yang ampuh untuk membuat sistem AI lebih mudah diakses, ia juga memiliki risiko dan keterbatasan tertentu. Pengguna kita mungkin mengembangkan harapan yang tidak realistis atau membentuk keterikatan emosional yang tidak sehat dengan sistem kita. Sebagai insinyur prompt dan pengembang, sangat penting untuk menyeimbangkan antara memanfaatkan keuntungan antropomorfisme dan memastikan bahwa pengguna mempertahankan pemahaman yang jelas tentang kemampuan dan keterbatasan AI.

Seiring berkembangnya bidang rekayasa prompt, kita dapat mengharapkan penyempurnaan dan inovasi lebih lanjut dalam cara kita berinteraksi dengan model bahasa besar. Namun, antropomorfisme sebagai sarana untuk memberikan pengalaman pengembang dan pengguna yang intuitif dan mudah diakses kemungkinan akan tetap menjadi prinsip fundamental dalam desain sistem-sistem ini.

Memisahkan Instruksi dari Data: Prinsip yang Sangat Penting

Penting untuk memahami prinsip fundamental yang mendasari keamanan dan keandalan sistem-sistem ini: pemisahan antara instruksi dan data.

Dalam ilmu komputer tradisional, perbedaan yang jelas antara data pasif dan instruksi aktif merupakan prinsip keamanan inti. Pemisahan ini membantu mencegah eksekusi kode yang tidak disengaja atau berbahaya yang dapat membahayakan integritas dan stabilitas sistem. Namun, LLM saat ini, yang utamanya dikembangkan sebagai model pengikut instruksi seperti chatbot, sering kali tidak memiliki pemisahan formal dan berprinsip ini.

Sejauh menyangkut LLM, instruksi dapat muncul di mana saja dalam input, baik itu dalam prompt sistem maupun prompt yang diberikan pengguna. Kurangnya pemisahan ini dapat menyebabkan potensi kerentanan dan perilaku yang tidak diinginkan, mirip dengan masalah yang dihadapi oleh basis data dengan injeksi SQL atau sistem operasi tanpa perlindungan memori yang tepat.

Saat Anda bekerja dengan LLM, sangat penting untuk menyadari keterbatasan ini dan mengambil langkah-langkah untuk mengurangi risikonya. Salah satu pendekatan adalah dengan hati-hati menyusun prompt dan input Anda untuk membedakan dengan jelas antara instruksi dan data. Metode umum untuk memberikan panduan eksplisit tentang apa yang merupakan instruksi dan apa yang harus diperlakukan sebagai data pasif melibatkan penandaan gaya markup. Prompt Anda dapat membantu LLM lebih baik memahami dan menghormati pemisahan ini.

Gambar 7. Menggunakan XML untuk membedakan antara instruksi, materi sumber, dan prompt pengguna

```
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>
```

Teknik lainnya adalah dengan menerapkan lapisan tambahan validasi dan sanitasi pada masukan yang diberikan kepada LLM. Dengan menyaring atau meloloskan setiap potensi instruksi atau potongan kode yang mungkin tertanam dalam data, Anda dapat mengurangi kemungkinan eksekusi yang tidak diinginkan. Pola seperti [Perantaraan Prompt](#) berguna untuk tujuan ini.

Selain itu, saat Anda merancang arsitektur aplikasi, pertimbangkan untuk memasukkan mekanisme yang menegakkan pemisahan instruksi dan data pada tingkat yang lebih tinggi. Ini bisa melibatkan penggunaan endpoint atau API terpisah untuk menangani instruksi dan data, menerapkan validasi dan penguraian masukan yang ketat, dan menerapkan *prinsip hak akses minimal* untuk membatasi ruang lingkup yang dapat diakses dan dieksekusi oleh LLM.

Prinsip Hak Akses Minimal

Menerapkan prinsip hak akses minimal itu seperti mengadakan pesta eksklusif di mana tamu hanya mendapat akses ke ruangan yang benar-benar mereka butuhkan. Bayangkan Anda sedang mengadakan pesta di sebuah mansion yang luas. Tidak semua orang perlu berkeliaran ke ruang penyimpanan anggur atau kamar utama, bukan? Dengan menerapkan prinsip ini, Anda pada dasarnya membagikan kunci yang hanya membuka pintu tertentu, memastikan bahwa setiap tamu, atau dalam kasus kita, setiap komponen aplikasi LLM Anda, hanya memiliki akses yang diperlukan untuk memenuhi perannya.

Ini bukan hanya soal pelit membagikan kunci, ini tentang mengakui bahwa di dunia di mana ancaman bisa datang dari mana saja, langkah cerdas adalah membatasi area bermain. Jika seseorang yang tidak diundang memaksa masuk ke pesta Anda, mereka akan mendapati diri mereka terkurung di ruang depan, yang secara drastis membatasi kekacauan yang bisa mereka perbuat. Jadi, saat mengamankan aplikasi LLM Anda, ingatlah: hanya berikan kunci ke ruangan yang diperlukan, dan jaga keamanan bagian mansion lainnya. Ini bukan sekadar soal tata krama yang baik; ini adalah keamanan yang baik.

Meskipun kondisi LLM saat ini mungkin belum memiliki pemisahan formal antara instruksi dan data, penting bagi Anda, sebagai pengembang, untuk memperhatikan keterbatasan ini dan mengambil langkah-langkah proaktif untuk mengurangi risiko. Dengan menerapkan praktik terbaik dari ilmu komputer tradisional dan mengadaptasinya dengan karakteristik unik LLM, Anda dapat membangun aplikasi yang lebih aman dan andal yang memanfaatkan kekuatan model-model ini sambil menjaga integritas sistem Anda.

Distilasi Prompt

Menyusun prompt yang sempurna seringkali merupakan tugas yang menantang dan memakan waktu, membutuhkan pemahaman mendalam tentang domain target dan

nuansa model bahasa. Di sinilah teknik “Distilasi Prompt” berperan, menawarkan pendekatan yang ampuh untuk rekayasa prompt yang memanfaatkan kemampuan model bahasa besar (LLM) untuk merampingkan dan mengoptimalkan proses.

Distilasi Prompt adalah teknik multi-tahap yang melibatkan penggunaan LLM untuk membantu dalam pembuatan, penyempurnaan, dan optimalisasi prompt. Alih-alih hanya mengandalkan keahlian dan intuisi manusia, pendekatan ini memanfaatkan pengetahuan dan kemampuan generatif LLM untuk bersama-sama menyusun prompt berkualitas tinggi.

Dengan terlibat dalam proses iteratif pembuatan, penyempurnaan, dan integrasi, Distilasi Prompt memungkinkan Anda membuat prompt yang lebih koheren, komprehensif, dan selaras dengan tugas atau output yang diinginkan. Perlu diperhatikan bahwa proses distilasi dapat dilakukan secara manual di salah satu dari banyak “playground” yang disediakan oleh vendor AI besar seperti OpenAI atau Anthropic, atau dapat diotomatiskan sebagai bagian dari kode aplikasi Anda, tergantung pada kasus penggunaannya.

Cara Kerjanya

Distilasi Prompt biasanya melibatkan langkah-langkah berikut:

1. **Identifikasi Maksud Utama:** Analisis prompt untuk menentukan tujuan utama dan hasil yang diinginkan. Hilangkan informasi yang tidak perlu dan fokus pada maksud utama dari prompt tersebut.
2. **Eliminasi Ambiguitas:** Tinjau prompt untuk mencari bahasa yang ambigu atau tidak jelas. Perjelas maknanya dan berikan detail spesifik untuk mengarahkan AI dalam menghasilkan respons yang akurat dan relevan.
3. **Sederhanakan Bahasa:** Sederhanakan prompt dengan menggunakan bahasa yang jelas dan ringkas. Hindari struktur kalimat yang kompleks, jargon, atau detail yang tidak perlu yang mungkin membingungkan AI atau menimbulkan gangguan.

4. **Berikan Konteks yang Relevan:** Sertakan hanya informasi kontekstual yang paling relevan yang diperlukan agar AI dapat memahami dan memproses prompt secara efektif. Hindari menyertakan detail yang tidak relevan atau berlebihan yang mungkin mengalihkan perhatian dari maksud utama.
5. **Iterasi dan Penyempurnaan:** Terus lakukan iterasi dan sempurnakan prompt berdasarkan respons dan umpan balik AI. Evaluasi output yang dihasilkan dan lakukan penyesuaian yang diperlukan untuk meningkatkan kejelasan dan efektivitas prompt. Secara opsional versi prompt Anda dalam basis data menggunakan **Prompt Object** untuk melacak iterasi dan memberi Anda kemampuan untuk dengan mudah membatalkan perubahan saat runtime.

Pembuatan Prompt Awal

Langkah pertama dalam Penyulingan Prompt adalah memberikan LLM (Model Bahasa Besar) sebuah deskripsi tingkat tinggi atau tujuan dari tugas atau keluaran yang diinginkan. Ini bisa berupa pernyataan singkat atau sekumpulan kata kunci yang menangkap esensi dari apa yang ingin dicapai oleh prompt tersebut. LLM kemudian menghasilkan prompt awal berdasarkan masukan ini, dengan memasukkan instruksi, contoh, atau konteks yang relevan yang dianggapnya sesuai.

Mari kita coba memberikan Claude 3 Opus prompt awal berikut:

```
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 merespons dengan keluaran berikut:

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.

Penyempurnaan Prompt

Setelah prompt awal dihasilkan, tinjau kembali dan minta LLM untuk mempertimbangkan konteks, batasan, atau umpan balik tambahan. Biarkan LLM menghasilkan versi prompt yang telah disempurnakan, dengan mempertimbangkan masukan yang diberikan dan pemahamannya sendiri tentang tugas tersebut. Proses penyempurnaan ini dapat diulang secara iteratif, dengan setiap iterasi semakin menyempurnakan prompt berdasarkan kolaborasi antara pengguna dan LLM.

Berikut adalah permintaan penyempurnaan saya, melanjutkan proses distilasi sebelumnya.

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.

Dan respons 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.

Kompresi Prompt

Seperti yang mungkin telah Anda perhatikan, LLM cenderung bertele-tele, kecuali jika Anda meminta mereka untuk ringkas. Prompt yang telah kita kerjakan di bagian sebelumnya cukup panjang, jadi mari kita minta Claude untuk mempersingkatnya:

- 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 merespons, dengan prompt yang jauh lebih singkat berikut:

- 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.

Direktif Sistem dan Integrasi Konteks

Selain menyempurnakan prompt itu sendiri, LLM juga dapat menghasilkan direktif sistem atau informasi konteks yang sesuai untuk mengarahkan keluaran akhir. Ketika merekayasa prompt AI yang akan diintegrasikan ke dalam kode aplikasi Anda, Anda hampir pasti akan fokus pada batasan keluaran pada tahap penyulingan ini, tetapi Anda juga dapat mengerjakan nada, gaya, format, atau parameter relevan lainnya yang memengaruhi respons yang dihasilkan.

Perakitan Prompt Akhir

Puncak dari proses Penyulingan Prompt adalah perakitan prompt akhir. Ini melibatkan penggabungan prompt yang telah disempurnakan, direktif sistem yang dihasilkan, dan konteks yang diintegrasikan menjadi kode yang kohesif dan komprehensif yang siap digunakan untuk menghasilkan keluaran yang diinginkan.



Anda dapat bereksperimen dengan kompresi prompt lagi pada tahap perakitan prompt akhir, dengan meminta LLM untuk menyusutkan kata-kata prompt menjadi rangkaian token terpendek yang mungkin sambil tetap mempertahankan esensi perilakunya. Ini tentu saja merupakan latihan yang hasilnya tidak pasti, tetapi khususnya dalam kasus prompt yang akan dijalankan dalam skala besar, peningkatan efisiensi dapat menghemat cukup banyak uang dalam konsumsi token.

Manfaat Utama

Dengan memanfaatkan pengetahuan dan kemampuan generatif LLM untuk menyempurnakan prompt Anda, prompt yang dihasilkan lebih mungkin terstruktur dengan baik, informatif, dan disesuaikan dengan tugas tertentu. Proses penyempurnaan iteratif membantu memastikan bahwa prompt berkualitas tinggi dan secara efektif menangkap maksud yang diinginkan. Manfaat lainnya meliputi:

Efisiensi dan Kecepatan: Penyulingan Prompt merampingkan proses rekayasa prompt dengan mengotomatisasi aspek-aspek tertentu dari pembuatan dan penyempurnaan prompt. Sifat kolaboratif dari teknik ini memungkinkan konvergensi yang lebih cepat menuju prompt yang efektif, mengurangi waktu dan upaya yang diperlukan untuk pembuatan prompt secara manual.

Konsistensi dan Skalabilitas: Penggunaan LLM dalam proses rekayasa prompt membantu menjaga konsistensi di seluruh prompt, karena LLM dapat mempelajari dan

menerapkan praktik terbaik dan pola dari prompt sukses sebelumnya. Konsistensi ini, dikombinasikan dengan kemampuan untuk menghasilkan prompt dalam skala besar, menjadikan Penyulingan Prompt sebagai teknik yang berharga untuk aplikasi berbasis AI skala besar.



Ide Proyek: Peralatan di tingkat pustaka yang menyederhanakan proses versi prompt dan penilaian dalam sistem yang melakukan penyulingan prompt otomatis sebagai bagian dari kode aplikasi mereka.

Untuk mengimplementasikan Penyulingan Prompt, pengembang dapat merancang alur kerja atau pipeline yang mengintegrasikan LLM pada berbagai tahap proses rekayasa prompt. Ini dapat dicapai melalui pemanggilan API, peralatan khusus, atau lingkungan pengembangan terintegrasi yang memfasilitasi interaksi yang mulus antara pengguna dan LLM selama pembuatan prompt. Detail implementasi spesifik dapat bervariasi tergantung pada platform LLM yang dipilih dan persyaratan aplikasi.

Bagaimana dengan fine-tuning?

Dalam buku ini, kita membahas prompt engineering dan RAG secara mendalam, tetapi tidak membahas fine-tuning. Alasan utama untuk keputusan ini adalah, menurut pendapat saya, sebagian besar pengembang aplikasi tidak membutuhkan fine-tuning untuk kebutuhan integrasi AI mereka.

Prompt engineering, yang melibatkan penyusunan prompt secara cermat dengan contoh zero-shot hingga few-shot, batasan, dan instruksi, dapat secara efektif mengarahkan model untuk menghasilkan respons yang relevan dan akurat untuk berbagai tugas. Dengan memberikan konteks yang jelas dan mempersempit jalur melalui prompt yang dirancang dengan baik, Anda dapat memanfaatkan pengetahuan luas dari model bahasa besar tanpa perlu melakukan fine-tuning.

Demikian pula, Retrieval-Augmented Generation (RAG) menawarkan pendekatan yang ampuh untuk mengintegrasikan AI ke dalam aplikasi. Dengan mengambil informasi yang relevan secara dinamis dari basis pengetahuan atau dokumen eksternal, RAG memberikan model konteks yang terfokus pada saat prompting. Hal ini memungkinkan model menghasilkan respons yang lebih akurat, terkini, dan spesifik domain, tanpa memerlukan proses fine-tuning yang memakan waktu dan sumber daya.

Meskipun fine-tuning dapat bermanfaat untuk domain atau tugas yang sangat khusus yang memerlukan tingkat kustomisasi yang mendalam, seringkali hal ini memerlukan biaya komputasi yang signifikan, persyaratan data, dan beban pemeliharaan. Untuk sebagian besar skenario pengembangan aplikasi, kombinasi prompt engineering dan RAG yang efektif sudah cukup untuk mencapai fungsionalitas dan pengalaman pengguna berbasis AI yang diinginkan.

Retrieval Augmented Generation (RAG)

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Apa itu Retrieval Augmented Generation?

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Bagaimana RAG Bekerja?

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Mengapa Menggunakan RAG dalam Aplikasi Anda?

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Mengimplementasikan RAG dalam Aplikasi Anda

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Persiapan Sumber Pengetahuan (Chunking)

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pemecahan Proposisi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Catatan Implementasi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pemeriksaan Kualitas

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Manfaat Pengambilan Berbasis Proposisi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh RAG dalam Dunia Nyata

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Studi Kasus: RAG dalam Aplikasi Persiapan Pajak Tanpa Embeddings

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Intelligent Query Optimization (IQO)

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Perangkingan Ulang

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Penilaian RAG (RAGAs)

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kesetiaan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Relevansi Jawaban

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Presisi Konteks

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Relevansi Konteks

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Penarikan Kembali Konteks

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Penarikan Kembali Entitas Konteks

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kemiripan Semantik Jawaban (ANSS)

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Ketepatan Jawaban

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kritik Aspek

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Tantangan dan Prospek Masa Depan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pemecahan Semantik: Meningkatkan Retrieval dengan Segmentasi Berbasis Konteks

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pengindeksan Hierarkis: Menstrukturkan Data untuk Retrieval yang Lebih Baik

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Self-RAG: Peningkatan Refleksi Diri

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

HyDE: Embedding Dokumen Hipotetis

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Apa itu Pembelajaran Kontrastif?

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Banyaknya Pekerja



Saya suka membayangkan komponen AI saya sebagai “pekerja” virtual yang hampir seperti manusia yang dapat diintegrasikan dengan mulus ke dalam logika aplikasi saya untuk melakukan tugas-tugas tertentu atau membuat keputusan kompleks. Idennya adalah untuk dengan sengaja memanusaiakan kemampuan LLM, sehingga tidak ada yang *terlalu* bersemangat dan memberikan mereka kualitas ajaib yang sebenarnya tidak mereka miliki.

Alih-alih hanya mengandalkan algoritma rumit atau implementasi manual yang memakan waktu, pengembang dapat mengkonseptualisasikan komponen AI sebagai entitas cerdas yang berdedikasi dan menyerupai manusia yang dapat dipanggil kapan pun diperlukan untuk mengatasi masalah kompleks dan memberikan solusi berdasarkan pelatihan dan pengetahuan mereka. Entitas-entitas ini tidak mudah teralihkan perhatiannya atau izin sakit. Mereka tidak secara spontan memutuskan untuk melakukan sesuatu dengan cara berbeda dari yang telah diinstruksikan kepada

mereka, dan secara umum, jika diprogram dengan benar, mereka juga tidak membuat kesalahan.

Dalam istilah teknis, prinsip utama di balik pendekatan ini adalah menguraikan tugas-tugas kompleks atau proses pengambilan keputusan menjadi unit-unit yang lebih kecil dan lebih mudah dikelola yang dapat ditangani oleh pekerja AI khusus. Setiap pekerja dirancang untuk fokus pada aspek tertentu dari masalah, membawa keahlian dan kemampuan uniknya ke dalam pekerjaan. Dengan mendistribusikan beban kerja di antara beberapa pekerja AI, aplikasi dapat mencapai efisiensi, skalabilitas, dan adaptabilitas yang lebih besar.

Sebagai contoh, pertimbangkan aplikasi web yang memerlukan moderasi waktu-nyata untuk konten buatan pengguna. Mengimplementasikan sistem moderasi yang komprehensif dari awal akan menjadi tugas yang menantang, membutuhkan upaya pengembangan yang signifikan dan pemeliharaan berkelanjutan. Namun, dengan menggunakan pendekatan Banyaknya Pekerja, pengembang dapat mengintegrasikan pekerja moderasi berbasis AI ke dalam logika aplikasi. Pekerja-pekerja ini dapat secara otomatis menganalisis dan menandai konten yang tidak pantas, membebaskan pengembang untuk fokus pada aspek kritis lainnya dari aplikasi.

Pekerja AI Sebagai Komponen Mandiri yang Dapat Digunakan Kembali

Aspek kunci dari pendekatan Banyaknya Pekerja adalah modularitasnya. Para pendukung pemrograman berorientasi objek telah memberi tahu kita selama beberapa dekade untuk memikirkan interaksi objek sebagai pesan. Nah, pekerja AI dapat dirancang sebagai komponen mandiri yang dapat digunakan kembali yang dapat “berbicara satu sama lain” melalui pesan dalam bahasa biasa, hampir seperti jika mereka benar-benar manusia kecil yang berbicara satu sama lain. Pendekatan yang longgar-terhubung ini memungkinkan aplikasi untuk beradaptasi dan berkembang seiring waktu, seiring

munculnya teknologi AI baru atau perubahan kebutuhan logika bisnis.

Dalam praktiknya, kebutuhan untuk merancang antarmuka dan protokol komunikasi yang jelas antar komponen tidak berubah meskipun pekerja AI terlibat di dalamnya. Anda tetap harus mempertimbangkan faktor-faktor lain seperti kinerja, skalabilitas, dan keamanan, tetapi sekarang ada “persyaratan lunak” yang sama sekali baru untuk dipertimbangkan juga. Misalnya, banyak pengguna keberatan jika data pribadi mereka digunakan untuk melatih model AI baru. Sudahkah Anda memverifikasi tingkat privasi yang disediakan oleh penyedia model yang Anda gunakan?

Pekerja AI Sebagai Mikroservis?

Saat Anda membaca tentang pendekatan Multitude of Workers (Kelompok Pekerja), Anda mungkin memperhatikan beberapa kesamaan dengan arsitektur Mikroservis. Keduanya menekankan pada penguraian sistem kompleks menjadi unit-unit yang lebih kecil, lebih mudah dikelola, dan dapat di-deploy secara independen. Sama seperti mikroservis yang dirancang untuk memiliki kopling longgar, berfokus pada kemampuan bisnis tertentu, dan berkomunikasi melalui API yang terdefinisi dengan baik, pekerja AI dirancang untuk bersifat modular, terspesialisasi dalam tugas mereka, dan berinteraksi satu sama lain melalui antarmuka dan protokol komunikasi yang jelas.

Namun, ada beberapa perbedaan kunci yang perlu diingat. Sementara mikroservis biasanya diimplementasikan sebagai proses atau layanan terpisah yang berjalan di mesin atau kontainer yang berbeda, pekerja AI dapat diimplementasikan sebagai komponen mandiri dalam satu aplikasi atau sebagai layanan terpisah, tergantung pada kebutuhan spesifik dan kebutuhan skalabilitas Anda. Selain itu, komunikasi antara pekerja AI sering melibatkan pertukaran informasi berbasis bahasa alami yang kaya, seperti prompts, instruksi, dan konten yang dihasilkan, alih-alih format data yang lebih terstruktur yang umumnya digunakan dalam mikroservis.

Terlepas dari perbedaan-perbedaan ini, prinsip-prinsip modularitas, kopling longgar, dan antarmuka komunikasi yang jelas tetap menjadi inti dari kedua pola tersebut. Dengan menerapkan prinsip-prinsip ini pada arsitektur pekerja AI Anda, Anda dapat menciptakan sistem yang fleksibel, dapat diskalakan, dan dapat dipelihara yang memanfaatkan kekuatan AI untuk memecahkan masalah kompleks dan memberikan nilai kepada pengguna Anda.

Pendekatan Multitude of Workers dapat diterapkan di berbagai domain dan aplikasi, memanfaatkan kekuatan AI untuk menangani tugas-tugas kompleks dan memberikan solusi cerdas. Mari kita jelajahi beberapa contoh konkret tentang bagaimana pekerja AI dapat digunakan dalam konteks yang berbeda.

Pengelolaan Akun

Hampir setiap aplikasi web mandiri memiliki konsep akun (atau pengguna). Di Olympia, kami menggunakan pekerja AI AccountManager yang diprogram untuk dapat menangani berbagai jenis permintaan perubahan yang terkait dengan akun pengguna.

Direktifnya berbunyi seperti ini:

```
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.
```

Keadaan awal akun yang dihasilkan oleh `account.to_directive` hanyalah deskripsi teks dari akun tersebut, termasuk data terkait yang relevan seperti pengguna, langganan, dan sebagainya.

Berbagai fungsi yang tersedia untuk `AccountManager` memberinya kemampuan untuk mengedit langganan pengguna, menambah dan menghapus konsultan AI dan berbagai jenis add-on berbayar lainnya, serta mengirim email pemberitahuan kepada pemilik akun. Selain fungsi `finished`, ia juga dapat `notify_human_administrator` jika menemui kesalahan selama pemrosesan atau membutuhkan bantuan lain terkait suatu permintaan.

Perhatikan bahwa jika ada pertanyaan, `AccountManager` dapat memilih untuk mencari di basis pengetahuan Olympia, di mana ia dapat menemukan instruksi tentang cara menangani kasus-kasus khusus dan situasi lain yang membuatnya tidak yakin bagaimana harus melanjutkan.

Aplikasi E-commerce

Dalam ranah e-commerce, pekerja AI dapat memainkan peran penting dalam meningkatkan pengalaman pengguna dan mengoptimalkan operasi bisnis. Berikut

beberapa cara pekerja AI dapat dimanfaatkan:

Rekomendasi Produk

Salah satu aplikasi pekerja AI yang paling powerful dalam e-commerce adalah menghasilkan rekomendasi produk yang dipersonalisasi. Dengan menganalisis perilaku pengguna, riwayat pembelian, dan preferensi, para pekerja ini dapat menyarankan produk yang disesuaikan dengan minat dan kebutuhan setiap pengguna individual.

Kunci untuk rekomendasi produk yang efektif adalah memanfaatkan kombinasi teknik penyaringan kolaboratif dan penyaringan berbasis konten. Penyaringan kolaboratif melihat perilaku pengguna yang serupa untuk mengidentifikasi pola dan membuat rekomendasi berdasarkan apa yang telah dibeli atau disukai oleh orang lain dengan selera serupa. Di sisi lain, penyaringan berbasis konten berfokus pada karakteristik dan atribut produk itu sendiri, merekomendasikan item yang memiliki fitur serupa dengan yang sebelumnya telah menarik minat pengguna.

Berikut contoh sederhana bagaimana Anda dapat mengimplementasikan pekerja rekomendasi produk dalam Ruby, kali ini menggunakan gaya pemrograman “[Railway Oriented \(ROP\)](#)” yang fungsional:

```
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
```



Gaya pemrograman fungsional Ruby yang digunakan dalam contoh ini dipengaruhi oleh F# dan Rust. Anda dapat membaca lebih lanjut tentang hal ini dalam [penjelasan teknik tersebut](#) oleh teman saya Chad Wooley di GitLab

Dalam contoh ini, `ProductRecommendationWorker` mengambil pengguna sebagai input dan menghasilkan rekomendasi produk yang dipersonalisasi dengan meneruskan objek nilai melalui rangkaian langkah fungsional. Mari kita uraikan setiap langkahnya:

1. `ValidateUser.validate`: Langkah ini memastikan bahwa pengguna valid dan memenuhi syarat untuk mendapatkan rekomendasi yang dipersonalisasi. Ini memeriksa apakah pengguna ada, aktif, dan memiliki data yang diperlukan untuk menghasilkan rekomendasi. Jika validasi gagal, hasil error dikembalikan, dan rantai dihentikan lebih awal.
2. `AnalyzeCurrentSession.analyze`: Jika pengguna valid, langkah ini menganalisis sesi penelusuran terkini pengguna untuk mengumpulkan informasi kontekstual. Ini melihat interaksi terbaru pengguna, seperti produk yang dilihat, kueri pencarian, dan isi keranjang belanja, untuk memahami minat dan maksud mereka saat ini.
3. `CollaborativeFilter.filter`: Dengan menggunakan *perilaku pengguna yang serupa*, langkah ini menerapkan teknik penyaringan kolaboratif untuk mengidentifikasi produk yang mungkin menarik bagi pengguna. Ini mempertimbangkan faktor-faktor seperti riwayat pembelian, penilaian, dan interaksi pengguna-produk untuk menghasilkan sekumpulan kandidat rekomendasi.

4. `ContentBasedFilter.filter`: Langkah ini lebih lanjut menyaring kandidat rekomendasi dengan menerapkan penyaringan berbasis konten. Ini membandingkan atribut dan karakteristik dari produk kandidat dengan *preferensi dan data historis pengguna* untuk memilih item yang paling relevan.
5. `ProductSelector.select`: Akhirnya, langkah ini memilih N produk teratas dari rekomendasi yang telah disaring berdasarkan kriteria yang telah ditentukan, seperti skor relevansi, popularitas, atau aturan bisnis lainnya. Produk yang terpilih kemudian dikembalikan sebagai rekomendasi personal final.

Keindahan menggunakan gaya pemrograman fungsional Ruby di sini adalah bahwa ini memungkinkan kita untuk merangkai langkah-langkah tersebut secara jelas dan ringkas. Setiap langkah berfokus pada tugas tertentu dan mengembalikan objek `Result`, yang bisa berupa sukses (`ok`) atau error (`err`). Jika ada langkah yang mengalami error, rantai akan dihentikan lebih awal, dan error tersebut diteruskan ke hasil akhir.

Pada pernyataan `case` di bagian akhir, kita melakukan pencocokan pola pada hasil akhir. Jika hasilnya adalah error (`ProductRecommendationError`), kita mencatat error tersebut menggunakan alat seperti Honeybadger untuk keperluan monitoring dan debugging. Jika hasilnya sukses (`ProductRecommendations`), kita menyiarkan event `:new_recommendations` menggunakan pustaka `pub/sub` Wisper, dengan meneruskan data pengguna dan rekomendasi yang telah dibuat.

Dengan memanfaatkan teknik pemrograman fungsional, kita dapat membuat worker rekomendasi produk yang modular dan mudah dipelihara. Setiap langkah bersifat mandiri dan dapat dengan mudah diuji, dimodifikasi, atau diganti tanpa mempengaruhi alur keseluruhan. Penggunaan pencocokan pola dan kelas `Result` membantu kita menangani error dengan baik dan memastikan worker gagal dengan cepat jika ada langkah yang mengalami masalah.

Tentu saja, ini adalah contoh yang disederhanakan, dan dalam skenario dunia nyata, Anda perlu melakukan integrasi dengan platform e-commerce Anda, menangani kasus-kasus khusus, dan bahkan mendalami implementasi algoritma rekomendasi. Namun,

prinsip utama dalam memecah masalah menjadi langkah-langkah yang lebih kecil dan memanfaatkan teknik pemrograman fungsional tetap sama.

Deteksi Penipuan

Berikut adalah contoh sederhana bagaimana Anda dapat mengimplementasikan worker deteksi penipuan menggunakan gaya Railway Oriented Programming (ROP) yang sama dalam 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
```

Kelas `FraudDetection` adalah sebuah *value object* yang mengenkapsulasi state deteksi penipuan untuk suatu transaksi tertentu. Kelas ini menyediakan cara yang terstruktur untuk menganalisis dan menilai risiko penipuan yang terkait dengan sebuah transaksi berdasarkan berbagai faktor risiko.

```
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
```

Kelas FraudDetection memiliki atribut-atribut berikut:

- `transaction`: Sebuah referensi ke transaksi yang sedang dianalisis untuk penipuan.
- `risk_factors`: Sebuah array yang menyimpan faktor-faktor risiko yang terkait dengan transaksi. Setiap faktor risiko direpresentasikan sebagai hash, di mana kunci adalah deskripsi dari faktor risiko, dan nilai adalah probabilitas penipuan yang terkait dengan faktor risiko tersebut.

Metode `add_risk_factor` memungkinkan penambahan faktor risiko ke array `risk_factors`. Metode ini menerima dua parameter: `description`, yang merupakan string yang mendeskripsikan faktor risiko, dan `probability`, yang merupakan float yang merepresentasikan probabilitas penipuan yang terkait dengan faktor risiko tersebut. Kita menggunakan kondisional `case..in` untuk melakukan pemeriksaan tipe sederhana.

Metode `high_risk?` yang akan diperiksa di akhir rantai adalah metode predikat yang membandingkan `fraud_probability` (dihitung dengan menjumlahkan probabilitas dari semua faktor risiko) dengan `RISK_THRESHOLD`.

Kelas `FraudDetection` menyediakan cara yang bersih dan terenkapsulasi untuk mengelola deteksi penipuan untuk sebuah transaksi. Kelas ini memungkinkan penambahan beberapa faktor risiko, masing-masing dengan deskripsi dan probabilitasnya sendiri, dan menyediakan metode untuk menentukan apakah transaksi dianggap berisiko tinggi berdasarkan probabilitas penipuan yang dihitung. Kelas ini dapat dengan mudah diintegrasikan ke dalam sistem deteksi penipuan yang lebih besar, di mana berbagai komponen dapat berkolaborasi untuk menilai dan mengurangi risiko transaksi penipuan.

Akhirnya, karena ini adalah buku tentang pemrograman menggunakan AI, berikut adalah contoh implementasi kelas `CheckCustomerHistory` yang memanfaatkan pemrosesan AI menggunakan modul `ChatCompletion` dari pustaka [Raix](#) saya:

```
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
```

Dalam contoh ini, `CheckCustomerHistory` mendefinisikan konstanta `INSTRUCTION` yang memberikan instruksi spesifik kepada model AI tentang cara menganalisis riwayat transaksi pelanggan untuk mencari indikator penipuan potensial melalui direktif sistem. Metode `self.check` adalah metode kelas yang menginisialisasi instance baru dari `CheckCustomerHistory` dengan objek `fraud_detection` dan memanggil metode `call` untuk melakukan analisis riwayat pelanggan.

Di dalam metode `call`, riwayat transaksi pelanggan diambil dan diformat menjadi transkrip yang diteruskan ke model AI. Model AI menganalisis riwayat transaksi berdasarkan instruksi yang diberikan dan mengembalikan ringkasan temuannya.

Temuan-temuan tersebut ditambahkan ke dalam objek `fraud_detection`, dan objek `fraud_detection` yang telah diperbarui dikembalikan sebagai `Result` yang berhasil.

Dengan memanfaatkan modul `ChatCompletion`, kelas `CheckCustomerHistory` dapat menggunakan kekuatan AI untuk menganalisis riwayat transaksi pelanggan dan mengidentifikasi indikator penipuan potensial. Hal ini memungkinkan teknik deteksi penipuan yang lebih canggih dan adaptif, karena model AI dapat belajar dan beradaptasi dengan pola dan anomali baru seiring waktu.

`FraudDetectionWorker` yang diperbarui dan kelas `CheckCustomerHistory` menunjukkan bagaimana pekerja AI dapat diintegrasikan dengan mulus, meningkatkan proses deteksi penipuan dengan kemampuan analisis dan pengambilan keputusan yang cerdas.

Analisis Sentimen Pelanggan

Berikut satu contoh serupa lagi tentang bagaimana Anda dapat mengimplementasikan pekerja analisis sentimen pelanggan. Kali ini dengan penjelasan yang lebih singkat, karena Anda seharusnya sudah mulai memahami cara kerja gaya pemrograman ini:

```
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
```

Dalam contoh ini, `CustomerSentimentAnalysisWorker` meliputi langkah-langkah seperti prapemrosesan umpan balik (misalnya, menghilangkan noise, tokenisasi), melakukan analisis sentimen untuk menentukan sentimen keseluruhan (positif, negatif, atau netral), mengekstrak frasa dan topik kunci, mengidentifikasi tren dan pola, serta menghasilkan wawasan yang dapat ditindaklanjuti berdasarkan analisis tersebut.

Aplikasi Kesehatan

Di bidang kesehatan, pekerja AI dapat membantu para profesional medis dan peneliti dalam berbagai tugas, yang mengarah pada peningkatan hasil pasien dan percepatan penemuan medis. Beberapa contohnya meliputi:

Penerimaan Pasien

Pekerja AI dapat mengefisienkan proses penerimaan pasien dengan mengotomatiskan berbagai tugas dan memberikan bantuan cerdas.

Penjadwalan Janji: Pekerja AI dapat menangani penjadwalan janji dengan memahami preferensi pasien, ketersediaan waktu, dan urgensi kebutuhan medis mereka. Mereka dapat berinteraksi dengan pasien melalui antarmuka percakapan, memandu mereka melalui proses penjadwalan dan menemukan slot janji yang paling sesuai berdasarkan kebutuhan pasien dan ketersediaan penyedia layanan kesehatan.

Pengumpulan Riwayat Medis: Selama penerimaan pasien, pekerja AI dapat membantu dalam mengumpulkan dan mendokumentasikan riwayat medis pasien. Mereka dapat terlibat dalam dialog interaktif dengan pasien, mengajukan pertanyaan yang relevan tentang kondisi medis sebelumnya, obat-obatan, alergi, dan riwayat kesehatan keluarga. Pekerja AI dapat menggunakan teknik pemrosesan bahasa alami untuk menginterpretasikan dan menyusun informasi yang dikumpulkan, memastikan informasi tersebut tercatat dengan akurat dalam rekam medis elektronik pasien.

Penilaian dan Stratifikasi Gejala: Pekerja AI dapat melakukan penilaian gejala awal dengan menanyakan pasien tentang gejala yang dialami saat ini, durasi, tingkat keparahan, dan faktor-faktor terkait. Dengan memanfaatkan basis pengetahuan medis dan model pembelajaran mesin, pekerja ini dapat menganalisis informasi yang diberikan dan menghasilkan diagnosis diferensial awal atau merekomendasikan langkah-langkah selanjutnya yang tepat, seperti menjadwalkan konsultasi dengan penyedia layanan kesehatan atau menyarankan langkah-langkah perawatan mandiri.

Verifikasi Asuransi: Pekerja AI dapat membantu dengan verifikasi asuransi selama penerimaan pasien. Mereka dapat mengumpulkan detail asuransi pasien, berkomunikasi dengan penyedia asuransi melalui API atau layanan web, dan memverifikasi kelayakan dan manfaat pertanggungan. Otomatisasi ini membantu mengefisienkan proses verifikasi asuransi, mengurangi beban administratif dan memastikan keakuratan informasi yang dicatat.

Edukasi dan Instruksi Pasien: Pekerja AI dapat menyediakan materi edukasi dan instruksi yang relevan kepada pasien berdasarkan kondisi medis atau prosedur yang akan datang. Mereka dapat memberikan konten yang dipersonalisasi, menjawab pertanyaan umum, dan memberikan panduan tentang persiapan pra-janji temu, instruksi pengobatan, atau perawatan pasca-pengobatan. Hal ini membantu pasien tetap terinformasi dan terlibat sepanjang perjalanan kesehatan mereka.

Dengan memanfaatkan pekerja AI dalam penerimaan pasien, organisasi layanan kesehatan dapat meningkatkan efisiensi, mengurangi waktu tunggu, dan meningkatkan pengalaman pasien secara keseluruhan. Pekerja-pekerja ini dapat menangani tugas-tugas rutin, mengumpulkan informasi yang akurat, dan memberikan bantuan yang dipersonalisasi, sehingga memungkinkan para profesional kesehatan untuk fokus memberikan perawatan berkualitas tinggi kepada pasien.

Penilaian Risiko Pasien

Pekerja AI dapat memainkan peran penting dalam menilai risiko pasien dengan menganalisis berbagai sumber data dan menerapkan teknik analitik tingkat lanjut.

Integrasi Data: Pekerja AI dapat mengumpulkan dan memahami data pasien dari berbagai sumber, seperti rekam medis elektronik (RME), pencitraan medis, hasil laboratorium, perangkat yang dapat dikenakan, dan determinan sosial kesehatan. Dengan menggabungkan informasi ini menjadi profil pasien yang komprehensif, pekerja AI dapat memberikan pandangan menyeluruh tentang status kesehatan dan faktor-faktor risiko pasien.

Stratifikasi Risiko: Pekerja AI dapat menggunakan model prediktif untuk melakukan stratifikasi pasien ke dalam berbagai kategori risiko berdasarkan karakteristik individual dan data kesehatan mereka. Stratifikasi risiko ini memungkinkan penyedia layanan kesehatan untuk memprioritaskan pasien yang memerlukan perhatian atau intervensi lebih segera. Sebagai contoh, pasien yang teridentifikasi berisiko tinggi untuk kondisi tertentu dapat ditandai untuk pemantauan lebih ketat, tindakan pencegahan, atau intervensi dini.

Profil Risiko Personal: Pekerja AI dapat menghasilkan profil risiko personal untuk setiap pasien, menyoroti faktor-faktor spesifik yang berkontribusi pada skor risiko mereka. Profil ini dapat mencakup wawasan tentang gaya hidup pasien, predisposisi genetik, faktor lingkungan, dan determinan sosial kesehatan. Dengan memberikan rincian lengkap tentang faktor-faktor risiko, pekerja AI dapat membantu penyedia layanan kesehatan menyesuaikan strategi pencegahan dan rencana perawatan sesuai kebutuhan individual pasien.

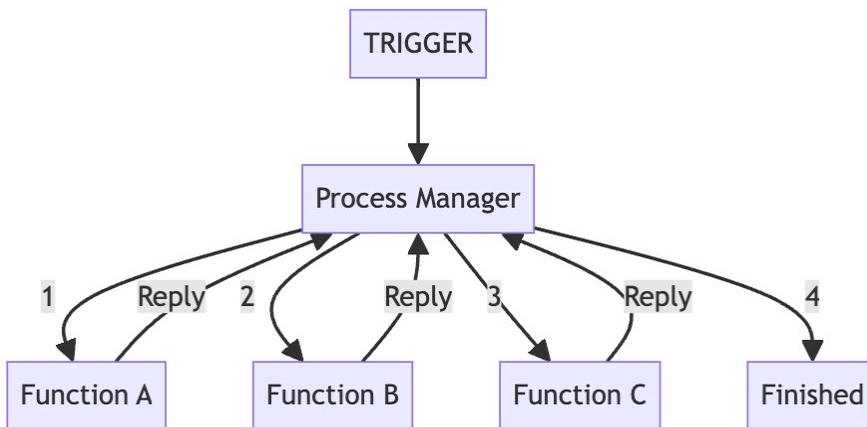
Pemantauan Risiko Berkelanjutan: Pekerja AI dapat terus memantau data pasien dan memperbarui penilaian risiko secara real-time. Ketika informasi baru tersedia, seperti perubahan tanda vital, hasil laboratorium, atau kepatuhan pengobatan, pekerja AI dapat menghitung ulang skor risiko dan memberitahu penyedia layanan kesehatan tentang perubahan signifikan apa pun. Pemantauan proaktif ini memungkinkan intervensi tepat waktu dan penyesuaian rencana perawatan pasien.

Dukungan Keputusan Klinis: Pekerja AI dapat mengintegrasikan hasil penilaian risiko ke dalam sistem dukungan keputusan klinis, memberikan rekomendasi dan peringatan berbasis bukti kepada penyedia layanan kesehatan. Misalnya, jika skor risiko pasien untuk kondisi tertentu melebihi ambang batas tertentu, pekerja AI dapat mengingatkan penyedia layanan kesehatan untuk mempertimbangkan tes diagnostik tertentu, tindakan pencegahan, atau pilihan pengobatan berdasarkan pedoman klinis dan praktik terbaik.

Pekerja-pekerja ini dapat memproses sejumlah besar data pasien, menerapkan analitik

canggih, dan menghasilkan wawasan yang dapat ditindaklanjuti untuk mendukung pengambilan keputusan klinis. Hal ini pada akhirnya mengarah pada peningkatan hasil pasien, pengurangan biaya kesehatan, dan peningkatan manajemen kesehatan populasi.

AI Worker sebagai Process Manager



Dalam konteks aplikasi berbasis AI, worker dapat dirancang untuk berfungsi sebagai Process Manager, seperti yang dijelaskan dalam buku “Enterprise Integration Patterns” oleh Gregor Hohpe. Process Manager adalah komponen pusat yang mempertahankan status proses dan menentukan langkah pemrosesan berikutnya berdasarkan hasil antara.

Ketika AI worker bertindak sebagai Process Manager, ia menerima pesan masuk yang menginisialisasi proses, yang dikenal sebagai *pesan pemicu*. AI worker kemudian mempertahankan status eksekusi proses (sebagai transkrip percakapan) dan menangani pesan melalui serangkaian langkah pemrosesan yang diimplementasikan sebagai fungsi alat, yang dapat berjalan secara berurutan atau paralel, dan dipanggil sesuai kebijakannya.



Jika Anda menggunakan kelas model AI seperti GPT-4 yang tahu cara mengeksekusi fungsi secara paralel, maka worker Anda dapat mengeksekusi beberapa langkah secara bersamaan. Sejujurnya, saya belum pernah mencobanya sendiri dan naluri saya mengatakan hasilnya mungkin bervariasi.

Setelah setiap langkah pemrosesan individual, kontrol dikembalikan ke AI worker, memungkinkannya menentukan langkah pemrosesan berikutnya berdasarkan status saat ini dan hasil yang diperoleh.

Simpan Pesan Pemicu Anda

Berdasarkan pengalaman saya, adalah bijaksana untuk mengimplementasikan pesan pemicu Anda sebagai objek berbasis basis data. Dengan cara itu, setiap instansi proses diidentifikasi oleh kunci primer yang unik dan memberi Anda tempat untuk menyimpan status yang terkait dengan eksekusi, termasuk transkrip percakapan AI.

Sebagai contoh, berikut adalah versi yang disederhanakan dari kelas model AccountChange Olympia, yang merepresentasikan permintaan untuk membuat perubahan pada akun pengguna.

```
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
```

Kelas `AccountChange` berfungsi sebagai pesan pemicu yang memulai proses untuk menangani permintaan perubahan akun. Perhatikan bagaimana pesan ini disiarkan ke subsistem pub/sub berbasis [Wisper](#) milik Olympia setelah transaksi pembuatan selesai dilakukan.

Menyimpan pesan pemicu dalam database seperti ini memberikan catatan permanen untuk setiap permintaan perubahan akun. Setiap instance dari kelas `AccountChange` diberi kunci primer yang unik, memungkinkan identifikasi dan pelacakan yang mudah untuk setiap permintaan individual. Hal ini sangat berguna untuk tujuan pencatatan audit, karena memungkinkan sistem untuk mempertahankan catatan historis dari semua perubahan akun, termasuk kapan perubahan diminta, perubahan apa yang diminta, dan status terkini dari setiap permintaan.

Dalam contoh yang diberikan, kelas `AccountChange` mencakup field seperti de-

`description` untuk menangkap detail dari perubahan yang diminta, `state` untuk merepresentasikan status terkini dari permintaan (misalnya, `requested`, `complete`, `requires_human_review`), dan `transcript` untuk menyimpan transkrip percakapan AI terkait permintaan tersebut. `field description` adalah prompt aktual yang digunakan untuk memulai chat completion pertama dengan AI. Penyimpanan data ini memberikan konteks yang berharga dan memungkinkan pelacakan dan analisis yang lebih baik terhadap proses perubahan akun.

Penyimpanan pesan pemicu dalam database memungkinkan penanganan kesalahan dan pemulihan yang kuat. Jika terjadi kesalahan selama pemrosesan permintaan perubahan akun, sistem akan menandai permintaan tersebut sebagai gagal dan mengalihkannya ke status yang memerlukan intervensi manusia. Hal ini memastikan bahwa tidak ada permintaan yang hilang atau terlupakan, dan setiap masalah dapat ditangani dan diselesaikan dengan tepat.

AI worker, sebagai Process Manager, menyediakan titik kontrol pusat dan memungkinkan kemampuan pelaporan dan debugging proses yang kuat. Namun, penting untuk dicatat bahwa menggunakan AI worker sebagai Process Manager untuk setiap skenario alur kerja dalam aplikasi Anda mungkin berlebihan.

Mengintegrasikan AI Worker ke Dalam Arsitektur Aplikasi Anda

Ketika mengintegrasikan AI worker ke dalam arsitektur aplikasi, beberapa pertimbangan teknis perlu diperhatikan untuk memastikan integrasi yang lancar dan komunikasi yang efektif antara AI worker dan komponen aplikasi lainnya. Bagian ini membahas aspek-aspek penting dalam merancang antarmuka tersebut, menangani aliran data, dan mengelola siklus hidup AI worker.

Merancang Antarmuka dan Protokol Komunikasi yang Jelas

Untuk memfasilitasi integrasi yang mulus antara AI worker dan komponen aplikasi lainnya, sangat penting untuk mendefinisikan antarmuka dan protokol komunikasi yang jelas. Pertimbangkan pendekatan-pendekatan berikut:

Integrasi Berbasis API: Mengekspos fungsionalitas pekerja AI melalui API yang terdefinisi dengan baik, seperti endpoint RESTful atau skema GraphQL. Hal ini memungkinkan komponen lain untuk berinteraksi dengan pekerja AI menggunakan permintaan dan respons HTTP standar. Integrasi berbasis API menyediakan kontrak yang jelas antara pekerja AI dan komponen yang menggunakannya, sehingga memudahkan pengembangan, pengujian, dan pemeliharaan titik-titik integrasi.

Komunikasi Berbasis Pesan: Mengimplementasikan pola komunikasi berbasis pesan, seperti antrian pesan atau sistem publikasi-langgan, untuk memungkinkan interaksi asinkron antara pekerja AI dan komponen lainnya. Pendekatan ini memisahkan pekerja AI dari bagian lain aplikasi, memungkinkan skalabilitas yang lebih baik, toleransi kesalahan, dan pengikatan yang longgar. Komunikasi berbasis pesan sangat berguna ketika pemrosesan yang dilakukan oleh pekerja AI memakan waktu atau membutuhkan banyak sumber daya, karena memungkinkan bagian lain dari aplikasi untuk terus berjalan tanpa menunggu pekerja AI menyelesaikan tugasnya.

Arsitektur Berbasis Event: Merancang sistem Anda di sekitar event dan pemicu yang mengaktifkan pekerja AI ketika kondisi tertentu terpenuhi. Pekerja AI dapat berlangganan event yang relevan dan bereaksi sesuai kebutuhan, melakukan tugas yang ditentukan ketika event terjadi. Arsitektur berbasis event memungkinkan pemrosesan real-time dan memungkinkan pekerja AI dipanggil sesuai kebutuhan, mengurangi konsumsi sumber daya yang tidak perlu. Pendekatan ini sangat cocok untuk skenario di mana pekerja AI perlu merespons tindakan atau perubahan tertentu dalam status aplikasi.

Menangani Aliran Data dan Sinkronisasi

Ketika mengintegrasikan pekerja AI ke dalam aplikasi Anda, sangat penting untuk memastikan aliran data yang lancar dan menjaga konsistensi data antara pekerja AI dan komponen lainnya. Pertimbangkan aspek-aspek berikut:

Persiapan Data: Sebelum memasukkan data ke dalam pekerja AI, Anda mungkin perlu melakukan berbagai tugas persiapan data, seperti pembersihan, pemformatan, dan/atau transformasi data masukan. Anda tidak hanya ingin memastikan bahwa pekerja AI dapat memproses secara efektif, tetapi juga ingin memastikan bahwa Anda tidak membuang token dengan memberikan perhatian pada informasi yang mungkin dianggap tidak berguna atau bahkan mengganggu oleh pekerja. Persiapan data mungkin melibatkan tugas-tugas seperti menghilangkan noise, menangani nilai yang hilang, atau mengkonversi tipe data.

Persistensi Data: Bagaimana Anda akan menyimpan dan mempertahankan data yang mengalir masuk dan keluar dari pekerja AI? Pertimbangkan faktor-faktor seperti volume data, pola query, dan skalabilitas. Apakah Anda perlu menyimpan transkrip AI sebagai refleksi dari “proses berpikirnya” untuk tujuan audit atau debugging, atau cukup hanya dengan menyimpan catatan hasilnya saja?

Pengambilan Data: Mendapatkan data yang dibutuhkan oleh worker mungkin melibatkan query ke database, membaca dari file, atau mengakses API eksternal. Pastikan untuk mempertimbangkan latensi dan bagaimana worker AI akan memiliki akses ke data yang paling mutakhir. Apakah mereka membutuhkan akses penuh ke database Anda atau haruskah Anda mendefinisikan cakupan akses mereka secara terbatas sesuai dengan apa yang mereka kerjakan? Bagaimana dengan skalabilitas? Pertimbangkan mekanisme caching untuk meningkatkan kinerja dan mengurangi beban pada sumber data yang mendasarinya.

Sinkronisasi Data: Ketika beberapa komponen, termasuk worker AI, mengakses dan memodifikasi data bersama, penting untuk menerapkan mekanisme sinkronisasi yang

tepat untuk menjaga konsistensi data. Strategi penguncian database, seperti optimistic atau pessimistic locking, dapat membantu Anda mencegah konflik dan memastikan integritas data. Terapkan teknik manajemen transaksi untuk mengelompokkan operasi data terkait dan mempertahankan properti ACID (Atomicity, Consistency, Isolation, dan Durability)

Penanganan Kesalahan dan Pemulihan: Terapkan mekanisme penanganan kesalahan dan pemulihan yang kuat untuk menangani masalah terkait data yang mungkin muncul selama proses aliran data. Tangani pengecualian dengan baik dan berikan pesan kesalahan yang bermakna untuk membantu debugging. Terapkan mekanisme retry dan strategi fallback untuk menangani kegagalan sementara atau gangguan jaringan. Tetapkan prosedur yang jelas untuk pemulihan dan pemulihan data jika terjadi kerusakan atau kehilangan data.

Dengan merancang dan menerapkan mekanisme aliran dan sinkronisasi data dengan hati-hati, Anda dapat memastikan bahwa worker AI Anda memiliki akses ke data yang akurat, konsisten, dan terbaru. Ini memungkinkan mereka untuk melakukan tugas mereka secara efektif dan menghasilkan hasil yang dapat diandalkan.

Mengelola Siklus Hidup Worker AI

Kembangkan proses standar untuk menginisialisasi dan mengkonfigurasi worker AI. Saya lebih suka kerangka kerja yang menstandarkan cara Anda mendefinisikan pengaturan seperti nama model, direktif sistem, dan definisi fungsi. Pastikan bahwa proses inisialisasi otomatis dan dapat direproduksi untuk memfasilitasi deployment dan scaling.

Terapkan mekanisme pemantauan dan pencatatan yang komprehensif untuk melacak kesehatan dan kinerja worker AI. Kumpulkan metrik seperti penggunaan sumber daya, waktu pemrosesan, tingkat kesalahan, dan throughput. Gunakan sistem pencatatan terpusat seperti ELK stack (Elasticsearch, Logstash, Kibana) untuk mengumpulkan dan menganalisis log dari beberapa worker AI.

Bangun toleransi kesalahan dan ketahanan ke dalam arsitektur pekerja AI. Terapkan mekanisme penanganan kesalahan dan pemulihan untuk mengatasi kegagalan atau pengecualian dengan baik. Model Bahasa Besar masih merupakan teknologi yang sangat baru; penyedia layanan sering mengalami gangguan pada waktu yang tidak terduga. Gunakan mekanisme pengulangan dan pemutus sirkuit untuk mencegah kegagalan berantai.

Kemampuan Komposisi dan Orkestrasi Pekerja AI

Salah satu keunggulan utama arsitektur pekerja AI adalah kemampuan komposisinya, yang memungkinkan Anda untuk menggabungkan dan mengorkestrasi beberapa pekerja AI untuk memecahkan masalah kompleks. Dengan memecah tugas yang lebih besar menjadi subtugas yang lebih kecil dan lebih mudah dikelola, yang masing-masing ditangani oleh pekerja AI khusus, Anda dapat menciptakan sistem yang kuat dan fleksibel. Dalam bagian ini, kita akan mengeksplorasi berbagai pendekatan untuk menyusun dan mengorkestrasi “sejumlah besar” pekerja AI.

Merantai Pekerja AI untuk Alur Kerja Bertahap

Dalam banyak skenario, tugas kompleks dapat diuraikan menjadi serangkaian langkah berurutan, di mana keluaran dari satu pekerja AI menjadi masukan untuk pekerja berikutnya. Perantaraan pekerja AI ini menciptakan alur kerja atau pipeline bertahap. Setiap pekerja AI dalam rantai berfokus pada subtugas tertentu, dan keluaran akhir adalah hasil dari upaya gabungan semua pekerja.

Mari kita lihat contoh dalam konteks aplikasi Ruby on Rails untuk memproses konten buatan pengguna. Alur kerja melibatkan langkah-langkah berikut, yang memang mungkin terlalu sederhana untuk layak diuraikan seperti ini dalam kasus penggunaan nyata, tetapi mereka membuat contoh lebih mudah dipahami:

1. **Pembersihan Teks:** Pekerja AI yang bertanggung jawab untuk menghapus tag HTML, mengubah teks menjadi huruf kecil, dan menangani normalisasi Unicode.
2. **Deteksi Bahasa:** Pekerja AI yang mengidentifikasi bahasa dari teks yang telah dibersihkan.
3. **Analisis Sentimen:** Pekerja AI yang menentukan sentimen (positif, negatif, atau netral) dari teks berdasarkan bahasa yang terdeteksi.
4. **Kategorisasi Konten:** Pekerja AI yang mengklasifikasikan teks ke dalam kategori yang telah ditentukan menggunakan teknik pemrosesan bahasa alami.

Berikut adalah contoh yang sangat disederhanakan tentang bagaimana Anda dapat merantai pekerja-pekerja AI ini menggunakan 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
```

Dalam contoh ini, kelas `ContentProcessor` diinisialisasi dengan teks mentah dan menghubungkan pekerja-pekerja AI bersama-sama dalam metode `process`. Setiap pekerja AI melakukan tugas spesifiknya dan meneruskan hasilnya ke pekerja berikutnya dalam rantai. Keluaran akhirnya adalah sebuah hash yang berisi teks yang telah dibersihkan, bahasa yang terdeteksi, sentimen, dan kategori konten.

Pemrosesan Paralel untuk Pekerja AI Independen

Dalam contoh sebelumnya, pekerja-pekerja AI dihubungkan secara sekuensial, di mana setiap pekerja memproses teks dan meneruskan hasilnya ke pekerja berikutnya. Namun, jika Anda memiliki beberapa pekerja AI yang dapat beroperasi secara independen pada input yang sama, Anda dapat mengoptimalkan alur kerja dengan memprosesnya secara paralel.

Dalam skenario yang diberikan, setelah pembersihan teks dilakukan oleh `TextCleanupWorker`, `LanguageDetectionWorker`, `SentimentAnalysisWorker`, dan `CategorizationWorker` semuanya dapat memproses teks yang telah dibersihkan secara independen. Dengan menjalankan pekerja-pekerja ini secara paralel, Anda dapat mengurangi waktu pemrosesan keseluruhan dan meningkatkan efisiensi alur kerja Anda.

Untuk mencapai pemrosesan paralel di Ruby, Anda dapat memanfaatkan teknik konkurensi seperti thread atau pemrograman asinkron. Berikut adalah contoh bagaimana Anda dapat memodifikasi kelas `ContentProcessor` untuk memproses tiga pekerja terakhir secara paralel menggunakan thread:

```
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
```

Dalam versi yang dioptimalkan ini, kita menggunakan pustaka `concurrent-ruby` untuk membuat objek `Concurrent::Future` untuk setiap pekerja AI independen. Sebuah `Future` merepresentasikan komputasi yang akan dijalankan secara asinkron dalam utas terpisah.

Setelah langkah pembersihan teks, kita membuat tiga objek `Future`: `language_future`, `sentiment_future`, dan `category_future`. Setiap `Future` menjalankan pekerja AI yang sesuai (`LanguageDetectionWorker`, `SentimentAnalysisWorker`, dan `CategorizationWorker`) dalam utas terpisah, dengan memberikan `cleaned_text` sebagai masukan.

Dengan memanggil metode `value` pada setiap `Future`, kita menunggu komputasi selesai dan mengambil hasilnya. Metode `value` memblokir hingga hasil tersedia, memastikan bahwa semua pekerja paralel telah selesai memproses sebelum melanjutkan.

Akhirnya, kita menyusun hash keluaran dengan teks yang telah dibersihkan dan hasil dari pekerja paralel, sama seperti dalam contoh aslinya.

Dengan memproses pekerja AI independen secara paralel, Anda dapat berpotensi mengurangi waktu pemrosesan keseluruhan dibandingkan dengan menjalankannya secara berurutan. Optimasi ini sangat bermanfaat ketika menangani tugas yang memakan

waktu atau ketika memproses data dalam jumlah besar.

Namun, penting untuk dicatat bahwa peningkatan kinerja yang sebenarnya bergantung pada berbagai faktor, seperti kompleksitas setiap pekerja, sumber daya sistem yang tersedia, dan overhead dari manajemen utas. Selalu menjadi praktik yang baik untuk melakukan pengujian kinerja dan profiling pada kode Anda untuk menentukan tingkat paralelisme yang optimal untuk kasus penggunaan spesifik Anda.

Selain itu, ketika mengimplementasikan pemrosesan paralel, perhatikan sumber daya bersama atau dependensi antar pekerja. Pastikan bahwa para pekerja dapat beroperasi secara independen tanpa konflik atau kondisi balapan. Jika ada dependensi atau sumber daya bersama, Anda mungkin perlu mengimplementasikan mekanisme sinkronisasi yang sesuai untuk menjaga integritas data dan menghindari masalah seperti kebuntuan atau hasil yang tidak konsisten.

Global Interpreter Lock (GIL) Ruby dan Pemrosesan Asinkron

Penting untuk memahami implikasi Global Interpreter Lock (GIL) Ruby ketika mempertimbangkan pemrosesan berbasis utas asinkron dalam Ruby.

GIL adalah mekanisme dalam interpreter Ruby yang memastikan hanya satu utas yang dapat mengeksekusi kode Ruby pada satu waktu, bahkan pada prosesor multi-core. Ini berarti bahwa meskipun beberapa utas dapat dibuat dan dikelola dalam sebuah proses Ruby, hanya satu utas yang dapat secara aktif mengeksekusi kode Ruby pada momen tertentu.

GIL dirancang untuk menyederhanakan implementasi interpreter Ruby dan menyediakan keamanan thread untuk struktur data internal Ruby. Namun, hal ini juga membatasi potensi eksekusi paralel yang sesungguhnya dari kode Ruby.

Ketika Anda menggunakan thread di Ruby, seperti dengan pustaka

`concurrent-ruby` atau kelas `Thread` bawaan, `thread-thread` tersebut tunduk pada batasan GIL. GIL memungkinkan setiap thread untuk mengeksekusi kode Ruby selama waktu yang singkat sebelum beralih ke thread lain, menciptakan ilusi eksekusi konkuren.

Namun, karena GIL, eksekusi kode Ruby yang sebenarnya tetap sekuensial. Saat satu thread mengeksekusi kode Ruby, thread lainnya pada dasarnya dijeda, menunggu giliran mereka untuk mendapatkan GIL dan mengeksekusi.

Ini berarti pemrosesan asinkron berbasis thread di Ruby paling efektif untuk tugas terikat I/O, seperti menunggu respons API eksternal (seperti model bahasa besar yang di-host pihak ketiga) atau melakukan operasi I/O file. Ketika sebuah thread menemui operasi I/O, thread tersebut dapat melepaskan GIL, memungkinkan thread lain untuk mengeksekusi sambil menunggu I/O selesai.

Di sisi lain, untuk tugas terikat CPU, seperti komputasi intensif atau pemrosesan pekerja AI yang berjalan lama, GIL dapat membatasi potensi peningkatan kinerja dari paralelisme berbasis thread. Karena hanya satu thread yang dapat mengeksekusi kode Ruby pada satu waktu, waktu eksekusi keseluruhan mungkin tidak berkurang secara signifikan dibandingkan dengan pemrosesan sekuensial.

Untuk mencapai eksekusi paralel yang sesungguhnya untuk tugas terikat CPU di Ruby, Anda mungkin perlu mengeksplorasi pendekatan alternatif, seperti:

- Menggunakan paralelisme berbasis proses dengan beberapa proses Ruby, masing-masing berjalan di core CPU yang terpisah.
- Memanfaatkan pustaka eksternal atau framework yang menyediakan ekstensi native atau antarmuka ke bahasa tanpa GIL, seperti C atau Rust.,
- Menggunakan framework komputasi terdistribusi atau antrian pesan untuk mendistribusikan tugas di beberapa mesin atau proses.

Sangat penting untuk mempertimbangkan sifat tugas Anda dan batasan yang diterapkan oleh GIL ketika merancang dan mengimplementasikan pemrosesan asinkron

di Ruby. Meskipun pemrosesan asinkron berbasis thread dapat memberikan manfaat untuk tugas terikat I/O, mungkin tidak menawarkan peningkatan kinerja yang signifikan untuk tugas terikat CPU karena batasan GIL.

Teknik Ensemble untuk Meningkatkan Akurasi

Teknik ensemble melibatkan penggabungan output dari beberapa pekerja AI untuk meningkatkan akurasi atau ketahanan sistem secara keseluruhan. Alih-alih mengandalkan satu pekerja AI, teknik ensemble memanfaatkan kecerdasan kolektif dari beberapa pekerja untuk membuat keputusan yang lebih terinformasi.



Ensemble sangat penting terutama jika berbagai bagian dari alur kerja Anda bekerja lebih baik dengan model AI yang berbeda-beda, hal yang lebih umum terjadi daripada yang Anda kira. Model-model yang kuat seperti GPT-4 sangat mahal dibandingkan dengan pilihan sumber terbuka yang kurang canggih, dan mungkin tidak diperlukan untuk setiap langkah alur kerja aplikasi Anda.

Salah satu teknik ensemble yang umum adalah pemungutan suara mayoritas, di mana beberapa pekerja AI secara independen memproses input yang sama, dan output akhir ditentukan berdasarkan konsensus mayoritas. Pendekatan ini dapat membantu mengurangi dampak kesalahan pekerja individual dan meningkatkan keandalan sistem secara keseluruhan.

Mari kita lihat contoh di mana kita memiliki tiga pekerja AI untuk analisis sentimen, masing-masing menggunakan model yang berbeda atau dilengkapi dengan konteks yang berbeda. Kita dapat menggabungkan output mereka menggunakan pemungutan suara mayoritas untuk menentukan prediksi sentimen akhir.

```
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
```

Dalam contoh ini, kelas `SentimentAnalysisEnsemble`. diinisialisasi dengan teks dan memanggil tiga pekerja AI analisis sentimen yang berbeda. Metode `analyze` mengumpulkan prediksi dari setiap pekerja dan menentukan sentimen mayoritas menggunakan metode `group_by` dan `max_by`. Keluaran akhirnya adalah sentimen yang menerima suara terbanyak dari ensembel pekerja



Ensembel jelas merupakan kasus di mana bereksperimen dengan paralelisme mungkin akan sangat bermanfaat.

Pemilihan dan Pemanggilan Dinamis Pekerja AI

Dalam beberapa atau bahkan kebanyakan kasus, pekerja AI tertentu yang akan dipanggil mungkin bergantung pada kondisi runtime atau input pengguna. Pemilihan dan pemanggilan dinamis pekerja AI memungkinkan fleksibilitas dan kemampuan adaptasi dalam sistem.



Anda mungkin akan tergoda untuk mencoba memasukkan banyak fungsionalitas ke dalam satu pekerja AI, memberikannya banyak fungsi dan prompt rumit yang menjelaskan cara memanggilnya. Tahanlah godaan itu, percayalah pada saya. Salah satu alasan mengapa pendekatan yang kita bahas dalam bab ini disebut “Multitude of Workers” adalah untuk mengingatkan kita bahwa sangat diinginkan untuk memiliki banyak pekerja khusus, masing-masing melakukan tugasnya sendiri untuk mencapai tujuan yang lebih besar.

Sebagai contoh, pertimbangkan aplikasi chatbot di mana pekerja AI yang berbeda bertanggung jawab untuk menangani berbagai jenis pertanyaan pengguna. Berdasarkan input pengguna, aplikasi secara dinamis memilih pekerja AI yang sesuai untuk memproses pertanyaan tersebut.

```
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
```

Dalam contoh ini, ChatbotController menerima query pengguna melalui aksi `process_query`. Pertama-tama ia menggunakan `QueryClassifierWorker` untuk

menentukan jenis query tersebut. Berdasarkan jenis query yang telah diklasifikasikan, controller secara dinamis memilih pekerja AI yang sesuai untuk menghasilkan respons. Pemilihan dinamis ini memungkinkan chatbot untuk menangani berbagai jenis query dan mengarahkannya ke pekerja AI yang relevan.



Karena pekerjaan `QueryClassifierWorker` relatif sederhana dan tidak memerlukan banyak konteks atau definisi fungsi, Anda mungkin bisa mengimplementasikannya menggunakan LLM kecil yang sangat cepat seperti `mistralai/mixtral-8x7b-instruct:nitro`. Model ini memiliki kemampuan yang hampir setara dengan level GPT-4 dalam banyak tugas dan, pada saat saya menulis ini, Groq dapat melayaninya dengan kecepatan luar biasa yaitu 444 token/detik.

Menggabungkan NLP Tradisional dengan LLM

Meskipun Model Bahasa Besar (LLM) telah merevolusi bidang pemrosesan bahasa natural (NLP), menawarkan keserbagunaan dan kinerja yang tak tertandingi dalam berbagai tugas, mereka tidak selalu menjadi solusi yang paling efisien atau hemat biaya untuk setiap masalah. Dalam banyak kasus, menggabungkan teknik NLP tradisional dengan LLM dapat menghasilkan pendekatan yang lebih optimal, terarah, dan ekonomis untuk menyelesaikan tantangan NLP tertentu.

Anggaplah LLM sebagai pisau Swiss Army dalam NLP—sangat serbaguna dan kuat, tetapi tidak selalu menjadi alat terbaik untuk setiap pekerjaan. Terkadang, alat khusus seperti pembuka tutup botol atau pembuka kaleng bisa lebih efektif dan efisien untuk tugas tertentu. Demikian pula, teknik NLP tradisional, seperti pengelompokan dokumen, identifikasi topik, dan klasifikasi, seringkali dapat memberikan solusi yang lebih terarah dan hemat biaya untuk aspek tertentu dalam alur kerja NLP Anda.

Salah satu keunggulan utama teknik NLP tradisional adalah efisiensi komputasinya. Metode-metode ini, yang sering mengandalkan model statistik yang lebih sederhana

atau pendekatan berbasis aturan, dapat memproses volume data teks yang besar dengan lebih cepat dan dengan beban komputasi yang lebih rendah dibandingkan dengan LLM. Hal ini membuat mereka sangat cocok untuk tugas-tugas yang melibatkan analisis dan pengorganisasian korpus dokumen yang besar, seperti pengelompokan artikel serupa atau mengidentifikasi topik utama dalam kumpulan teks.

Selain itu, teknik NLP tradisional seringkali dapat mencapai akurasi dan presisi tinggi untuk tugas-tugas tertentu, terutama ketika dilatih pada dataset khusus domain. Sebagai contoh, pengklasifikasi dokumen yang disetel dengan baik menggunakan algoritma pembelajaran mesin tradisional seperti Support Vector Machines (SVM) atau Naive Bayes dapat mengkategorikan dokumen ke dalam kategori yang telah ditentukan secara akurat dengan biaya komputasi minimal.

Namun, LLMs benar-benar unggul ketika berhadapan dengan tugas-tugas yang memerlukan pemahaman yang lebih dalam tentang bahasa, konteks, dan penalaran. Kemampuan mereka untuk menghasilkan teks yang koheren dan relevan secara kontekstual, menjawab pertanyaan, dan meringkas bagian-bagian panjang tidak dapat ditandingi oleh metode NLP tradisional. LLMs dapat menangani fenomena linguistik yang kompleks secara efektif, seperti ambiguitas, koreferensi, dan ungkapan idiomatik, menjadikannya sangat berharga untuk tugas-tugas yang memerlukan generasi atau pemahaman bahasa alami.

Kekuatan sesungguhnya terletak pada penggabungan teknik NLP tradisional dengan LLMs untuk menciptakan pendekatan hibrid yang memanfaatkan kekuatan keduanya. Dengan menggunakan metode NLP tradisional untuk tugas-tugas seperti praproses dokumen, pengelompokan, dan ekstraksi topik, Anda dapat mengatur dan menyusun data teks Anda secara efisien. Informasi terstruktur ini kemudian dapat diumpangkan ke LLMs untuk tugas-tugas yang lebih canggih, seperti menghasilkan ringkasan, menjawab pertanyaan, atau membuat laporan komprehensif.

Misalnya, mari kita pertimbangkan kasus penggunaan di mana Anda ingin menghasilkan laporan tren untuk domain tertentu berdasarkan korpus besar dokumen tren

individual. Alih-alih hanya mengandalkan LLMs, yang bisa mahal secara komputasi dan memakan waktu untuk memproses volume teks yang besar, Anda dapat menggunakan pendekatan hibrid:

1. Gunakan teknik NLP tradisional, seperti pemodelan topik (misalnya, Latent Dirichlet Allocation) atau algoritma pengelompokan (misalnya, K-means), untuk mengelompokkan dokumen tren yang serupa dan mengidentifikasi tema dan topik utama dalam korpus.
2. Umpangkan dokumen yang telah dikelompokkan dan topik yang telah diidentifikasi ke LLM, memanfaatkan kemampuan pemahaman dan generasi bahasanya yang unggul untuk membuat ringkasan yang koheren dan informatif untuk setiap kelompok atau topik.
3. Terakhir, gunakan LLM untuk menghasilkan laporan tren yang komprehensif dengan menggabungkan ringkasan individual, menyoroti tren yang paling signifikan, dan memberikan wawasan serta rekomendasi berdasarkan informasi yang telah dikumpulkan.

Dengan menggabungkan teknik NLP tradisional dengan LLMs dengan cara ini, Anda dapat memproses sejumlah besar data teks secara efisien, mengekstrak wawasan yang bermakna, dan menghasilkan laporan berkualitas tinggi sambil mengoptimalkan sumber daya dan biaya komputasi.

Saat Anda mulai menjalankan proyek-proyek NLP, sangat penting untuk mengevaluasi dengan cermat persyaratan dan batasan spesifik dari setiap tugas dan mempertimbangkan bagaimana metode NLP tradisional dan LLM dapat dimanfaatkan bersama untuk mencapai hasil terbaik. Dengan menggabungkan efisiensi dan presisi dari teknik tradisional dengan keluwesan dan kekuatan LLM, Anda dapat menciptakan solusi NLP yang sangat efektif dan ekonomis yang memberikan nilai bagi pengguna dan para stakeholder Anda.

Penggunaan Alat



Dalam ranah pengembangan aplikasi berbasis AI, konsep “penggunaan alat” atau “pemanggilan fungsi” telah muncul sebagai teknik yang ampuh yang memungkinkan LLM Anda untuk terhubung dengan alat eksternal, API, fungsi, basis data, dan sumber daya lainnya. Pendekatan ini memungkinkan serangkaian perilaku yang lebih kaya daripada sekadar menghasilkan teks, dan interaksi yang lebih dinamis antara komponen AI Anda dan seluruh ekosistem aplikasi Anda. Seperti yang akan kita bahas dalam bab ini, penggunaan alat juga memberi Anda opsi untuk membuat model AI Anda menghasilkan data dengan cara yang terstruktur.

Apa itu Penggunaan Alat?

Penggunaan alat, yang juga dikenal sebagai pemanggilan fungsi, adalah teknik yang memungkinkan pengembang untuk menentukan daftar fungsi yang dapat diinterak-

sikan oleh LLM selama proses generasi. Alat-alat ini dapat berkisar dari fungsi utilitas sederhana hingga API atau kueri basis data yang kompleks. Dengan memberikan LLM akses ke alat-alat ini, pengembang dapat memperluas kemampuan model dan memungkinkannya untuk melakukan tugas-tugas yang memerlukan pengetahuan atau tindakan eksternal.

Gambar 8. Contoh definisi fungsi untuk pekerja AI yang menganalisis dokumen

```
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  }
27 }.freeze
```

Ide utama di balik penggunaan alat adalah memberikan LLM kemampuan untuk secara dinamis memilih dan mengeksekusi alat yang sesuai berdasarkan masukan pengguna atau tugas yang sedang dikerjakan. Alih-alih hanya mengandalkan pengetahuan

yang telah dilatih sebelumnya pada model, penggunaan alat memungkinkan LLM untuk memanfaatkan sumber daya eksternal untuk menghasilkan respons yang lebih akurat, relevan, dan dapat ditindaklanjuti. Penggunaan alat membuat teknik seperti RAG (Generasi yang Diperkaya Pengambilan) jauh lebih mudah diimplementasikan dibandingkan cara lainnya.

Perlu diperhatikan bahwa kecuali dinyatakan lain, buku ini mengasumsikan model AI Anda tidak memiliki akses ke alat bawaan sisi server. Setiap alat yang ingin Anda sediakan untuk AI Anda harus dideklarasikan secara eksplisit oleh Anda dalam setiap permintaan API, dengan ketentuan untuk mengirimkannya jika dan ketika AI Anda memberi tahu Anda bahwa ia ingin menggunakan alat tersebut dalam responnya.

Potensi Penggunaan Alat

Penggunaan alat membuka berbagai kemungkinan untuk aplikasi berbasis AI. Berikut beberapa contoh yang dapat dicapai dengan penggunaan alat:

1. **Chatbot dan Asisten Virtual:** Dengan menghubungkan LLM ke alat eksternal, chatbot dan asisten virtual dapat melakukan tugas yang lebih kompleks, seperti mengambil informasi dari basis data, melakukan pemanggilan API, atau berinteraksi dengan sistem lain. Misalnya, chatbot dapat menggunakan alat CRM untuk mengubah status kesepakatan berdasarkan permintaan pengguna.
2. **Analisis Data dan Wawasan:** LLM dapat dihubungkan ke alat atau pustaka analisis data untuk melakukan tugas pemrosesan data tingkat lanjut. Ini memungkinkan aplikasi untuk menghasilkan wawasan, melakukan analisis komparatif, atau memberikan rekomendasi berbasis data berdasarkan pertanyaan pengguna.

3. **Pencarian dan Pengambilan Informasi:** Penggunaan alat memungkinkan LLM untuk berinteraksi dengan mesin pencari, basis data vektor, atau sistem pengambilan informasi lainnya. Dengan mengubah pertanyaan pengguna menjadi kueri pencarian, LLM dapat mengambil informasi yang relevan dari berbagai sumber dan memberikan jawaban komprehensif atas pertanyaan pengguna.
4. **Integrasi dengan Layanan Eksternal:** Penggunaan alat memungkinkan integrasi yang mulus antara aplikasi berbasis AI dan layanan atau API eksternal. Misalnya, LLM dapat berinteraksi dengan API cuaca untuk memberikan pembaruan cuaca real-time atau API penerjemahan untuk menghasilkan respons dalam berbagai bahasa.

Alur Kerja Penggunaan Alat

Alur kerja penggunaan alat biasanya melibatkan empat langkah utama:

1. Menyertakan definisi fungsi dalam konteks permintaan Anda
2. Pemilihan alat secara dinamis (atau eksplisit)
3. Eksekusi fungsi
4. Kelanjutan opsional dari prompt awal

Mari kita tinjau setiap langkah ini secara detail.

Menyertakan definisi fungsi dalam konteks permintaan Anda

AI mengetahui alat apa saja yang tersedia karena Anda memberikan daftar sebagai bagian dari permintaan penyelesaian Anda (biasanya didefinisikan sebagai fungsi menggunakan varian skema JSON).

Sintaks yang tepat untuk definisi alat bergantung pada model yang digunakan.

Beginilah cara Anda mendefinisikan fungsi `get_weather` di 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 }
```

Dan beginilah cara Anda mendefinisikan fungsi yang sama untuk GPT-4, dengan meneruskannya sebagai nilai parameter 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 }
```

Hampir sama, hanya berbeda tanpa alasan yang jelas! Sungguh menjengkelkan.

Definisi fungsi menentukan nama, deskripsi, dan parameter masukan. Parameter masukan dapat didefinisikan lebih lanjut menggunakan atribut seperti enums untuk membatasi nilai yang dapat diterima, dan menentukan apakah suatu parameter wajib atau tidak.

Selain definisi fungsi yang sebenarnya, Anda juga dapat menyertakan instruksi atau konteks tentang mengapa dan bagaimana menggunakan fungsi tersebut dalam direktif sistem.

Sebagai contoh, perangkat Web Search saya di Olympia menyertakan direktif sistem ini, yang mengingatkan AI bahwa ia memiliki perangkat-perangkat yang disebutkan tersebut:

```
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.
```

Memberikan deskripsi yang detail dianggap sebagai faktor terpenting dalam kinerja alat. Deskripsi Anda harus menjelaskan setiap detail tentang alat tersebut, termasuk:

- Apa yang dilakukan oleh alat tersebut
- Kapan alat tersebut sebaiknya digunakan (dan kapan tidak)
- Apa arti setiap parameter dan bagaimana pengaruhnya terhadap perilaku alat

- Hal-hal penting yang perlu diperhatikan atau batasan yang berlaku pada implementasi alat

Semakin banyak konteks yang dapat Anda berikan kepada AI tentang alat-alat Anda, semakin baik kemampuannya dalam menentukan kapan dan bagaimana menggunakannya. Misalnya, Anthropic merekomendasikan setidaknya 3-4 kalimat per deskripsi alat untuk seri Claude 3, lebih banyak jika alatnya kompleks.

Mungkin tidak terasa intuitif, tetapi deskripsi juga dianggap lebih penting daripada contoh. Meskipun Anda dapat menyertakan contoh cara menggunakan alat dalam deskripsinya atau dalam prompt yang menyertainya, ini kurang penting dibandingkan dengan penjelasan yang jelas dan komprehensif tentang tujuan dan parameter alat tersebut. Tambahkan contoh hanya setelah Anda telah melengkapi deskripsinya secara menyeluruh.

Berikut adalah contoh spesifikasi fungsi API bergaya 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             "required": [ "type", "number" ]
46         }
47     },
48     "required": [ "name", "email", "identification" ]
49 }
50 }
51 }
```



Dalam praktiknya, beberapa model mengalami kesulitan menangani spesifikasi fungsi bersarang dan menangani tipe data keluaran yang kompleks seperti arrays, dictionaries dan sebagainya. Namun secara teori, Anda seharusnya bisa memberikan spesifikasi JSON Schema dengan kedalaman yang tak terbatas!

Pemilihan Alat Dinamis

Ketika Anda menjalankan chat completion yang menyertakan definisi alat, LLM secara dinamis memilih alat yang paling sesuai untuk digunakan dan menghasilkan parameter masukan yang diperlukan untuk setiap alat.

Dalam praktiknya, kemampuan AI untuk memanggil fungsi yang *tepat*, dan *tepat* mengikuti spesifikasi Anda untuk masukan bersifat tidak pasti. Menurunkan parameter temperature hingga 0.0 sangat membantu, tetapi berdasarkan pengalaman saya, Anda masih akan menemui kesalahan sesekali. Kegagalan tersebut termasuk nama fungsi yang dihalusinasi, parameter masukan yang salah nama atau bahkan hilang. Parameter dikirimkan sebagai JSON, yang berarti terkadang Anda akan melihat kesalahan yang disebabkan oleh JSON yang terpotong, salah kutip, atau rusak.



Pola [Self Healing Data](#) dapat membantu [memperbaiki secara otomatis](#) pemanggilan fungsi yang rusak karena kesalahan sintaks.

Pemilihan Alat Paksa (alias Eksplisit)

Beberapa model memberi Anda opsi untuk memaksa pemanggilan fungsi tertentu, sebagai parameter dalam permintaan. Jika tidak, keputusan untuk memanggil fungsi sepenuhnya bergantung pada kebijaksanaan AI.

Kemampuan untuk memaksa pemanggilan fungsi sangat penting dalam skenario tertentu di mana Anda ingin memastikan bahwa alat atau fungsi tertentu dijalankan, terlepas dari proses pemilihan dinamis AI. Ada beberapa alasan mengapa kemampuan ini penting:

1. **Kontrol Eksplisit:** Anda mungkin menggunakan AI sebagai *Komponen Diskrit* atau dalam alur kerja yang telah ditentukan yang mengharuskan eksekusi fungsi tertentu pada waktu tertentu. Dengan memaksa pemanggilan, Anda dapat

menjamin bahwa fungsi yang diinginkan dipanggil alih-alih harus meminta AI dengan sopan untuk melakukannya.

2. **Debugging dan Pengujian:** Saat mengembangkan dan menguji aplikasi berbasis AI, kemampuan untuk memaksa pemanggilan fungsi sangat berharga untuk tujuan debugging. Dengan memicu fungsi tertentu secara eksplisit, Anda dapat mengisolasi dan menguji komponen individual dari aplikasi Anda. Ini memungkinkan Anda untuk memverifikasi kebenaran implementasi fungsi, memvalidasi parameter masukan, dan memastikan bahwa hasil yang diharapkan dikembalikan.
3. **Menangani Kasus Tepi:** Mungkin ada kasus tepi atau skenario khusus di mana proses pemilihan dinamis AI mungkin tidak memilih untuk mengeksekusi fungsi yang seharusnya, dan Anda mengetahui hal ini berdasarkan proses eksternal. Dalam kasus seperti ini, kemampuan untuk memaksa pemanggilan fungsi memungkinkan Anda menangani situasi tersebut secara eksplisit. Tentukan aturan atau kondisi dalam logika aplikasi Anda untuk menentukan kapan harus mengesampingkan kebijaksanaan AI.
4. **Konsistensi dan Reprodusibilitas:** Jika Anda memiliki urutan fungsi tertentu yang perlu dieksekusi dalam urutan tertentu, memaksa pemanggilan menjamin bahwa urutan yang sama diikuti setiap kali. Ini sangat penting dalam aplikasi di mana konsistensi dan perilaku yang dapat diprediksi sangat kritis, seperti dalam sistem keuangan atau simulasi ilmiah.
5. **Optimasi Kinerja:** Dalam beberapa kasus, memaksa pemanggilan fungsi dapat menghasilkan optimasi kinerja. Jika Anda tahu bahwa fungsi tertentu diperlukan untuk tugas tertentu dan proses pemilihan dinamis AI mungkin menimbulkan overhead yang tidak perlu, Anda dapat melewati proses pemilihan dan langsung memanggil fungsi yang diperlukan. Ini dapat membantu mengurangi latensi dan meningkatkan efisiensi keseluruhan aplikasi Anda.

Singkatnya, kemampuan untuk memaksa pemanggilan fungsi dalam aplikasi berbasis AI memberikan kontrol eksplisit, membantu dalam debugging dan pengujian, menangani

kasus tepi, memastikan konsistensi dan reproduksibilitas. Ini adalah alat yang kuat dalam arsenal Anda, tetapi kita perlu membahas satu aspek lagi dari fitur penting ini.



Dalam banyak kasus pengambilan keputusan, kita selalu ingin model melakukan pemanggilan fungsi dan mungkin tidak pernah ingin model merespons hanya dengan pengetahuan internalnya. Misalnya, jika Anda merutekan antara beberapa model yang khusus untuk tugas yang berbeda (input multibahasa, matematika, dll), Anda mungkin menggunakan model pemanggilan fungsi untuk mendelegasikan permintaan ke salah satu model pembantu dan tidak pernah merespons secara independen.

Parameter Pilihan Alat

GPT-4 dan model bahasa lain yang mendukung pemanggilan fungsi memberi Anda parameter `tool_choi ce` untuk mengontrol apakah penggunaan alat diperlukan sebagai bagian dari penyelesaian. Parameter ini memiliki tiga nilai yang mungkin:

- `auto` memberi AI keleluasaan penuh untuk menggunakan alat atau sekadar merespons
- `required` memberi tahu AI bahwa ia *harus* memanggil alat *alih-alih* merespons, tetapi membiarkan pemilihan alat terserah pada AI
- Opsi ketiga adalah menetapkan parameter dari `name_of_function` yang ingin Anda paksa. Lebih lanjut tentang itu di bagian berikutnya.



Perhatikan bahwa jika Anda mengatur pilihan alat ke `required`, model akan dipaksa untuk memilih fungsi yang paling relevan untuk dipanggil dari fungsi-fungsi yang disediakan, bahkan jika tidak ada yang benar-benar cocok dengan prompt tersebut. Pada saat publikasi, saya tidak mengetahui adanya model yang akan mengembalikan `respons tool_calls` kosong, atau menggunakan cara lain untuk memberi tahu Anda bahwa model tersebut tidak menemukan fungsi yang sesuai untuk dipanggil.

Memaksa Fungsi Untuk Mendapatkan Output Terstruktur

Kemampuan untuk memaksa pemanggilan fungsi memberi Anda cara untuk memaksa data terstruktur keluar dari chat completion alih-alih harus mengekstraknya sendiri dari respons teks biasa.

Mengapa memaksa fungsi untuk mendapatkan output terstruktur itu penting? Singkatnya, karena ekstraksi data terstruktur dari output LLM itu merepotkan. Anda bisa membuat hidup Anda sedikit lebih mudah dengan meminta data dalam format XML, tapi kemudian Anda harus mengurai XML tersebut. Dan apa yang Anda lakukan ketika XML itu hilang karena AI Anda merespons: “Maaf, saya tidak dapat menghasilkan data yang Anda minta karena bla, bla, bla...”

Ketika menggunakan alat dengan cara ini:

- Anda sebaiknya mendefinisikan satu alat saja dalam permintaan Anda
- Ingat untuk memaksa penggunaan fungsinya menggunakan parameter `tool_choice`.
- Ingat bahwa model akan meneruskan input ke alat tersebut, jadi nama alat dan deskripsinya harus dari perspektif model, bukan perspektif Anda.

Poin terakhir ini membutuhkan contoh untuk kejelasan. Misalkan Anda meminta AI untuk melakukan analisis sentimen pada teks pengguna. Nama fungsinya tidak akan menjadi `analyze_sentiment`, tetapi lebih tepatnya akan menjadi seperti `save_sentiment_analysis`. AI-lah yang melakukan analisis sentimen, *bukan alatnya*. Yang dilakukan alat (dari perspektif AI) hanyalah menyimpan hasil analisis.

Berikut adalah contoh penggunaan Claude 3 untuk merekam ringkasan gambar ke dalam JSON yang terstruktur dengan baik, kali ini dari command line menggunakan

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      "name": "record_summary",  
12      "description": "Record summary of image into well-structured JSON.",  
13      "input_schema": {  
14        "type": "object",  
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 }'

```

Dalam contoh yang diberikan, kita menggunakan model Claude 3 dari Anthropic untuk

menghasilkan ringkasan JSON terstruktur dari sebuah gambar. Berikut cara kerjanya:

1. Kita mendefinisikan satu alat bernama `record_summary` dalam array `tools` pada muatan permintaan. Alat ini bertanggung jawab untuk merekam ringkasan gambar ke dalam JSON yang terstruktur dengan baik.
2. Alat `record_summary` memiliki `input_schema` yang menentukan struktur output JSON yang diharapkan. Ini mendefinisikan tiga properti:
 - `key_colors`: Sebuah array objek yang merepresentasikan warna-warna utama dalam gambar. Setiap objek warna memiliki properti untuk nilai merah, hijau, dan biru (berkisar dari 0.0 hingga 1.0) dan nama warna yang mudah dibaca manusia dalam format `snake_case`.
 - `description`: Properti string untuk deskripsi singkat gambar, dibatasi hingga 1-2 kalimat.
 - `estimated_year`: Properti integer opsional untuk perkiraan tahun gambar diambil, jika tampak sebagai foto non-fiksi.
3. Dalam array `messages`, kita menyediakan data gambar sebagai string base64-encoded beserta tipe medianya. Ini memungkinkan model untuk memproses gambar sebagai bagian dari input.
4. Kita juga mengarahkan Claude untuk menggunakan alat `record_summary` untuk mendeskripsikan gambar.
5. Ketika permintaan dikirim ke model Claude 3, ia menganalisis gambar dan menghasilkan ringkasan JSON berdasarkan `input_schema` yang ditentukan. Model mengekstrak warna-warna utama, memberikan deskripsi singkat, dan memperkirakan tahun gambar diambil (jika berlaku).
6. Ringkasan JSON yang dihasilkan diteruskan sebagai parameter ke alat `record_summary`, memberikan representasi terstruktur dari karakteristik utama gambar.

Dengan menggunakan alat `record_summary` dengan `input_schema` yang terdefinisi dengan baik, kita dapat memperoleh ringkasan JSON terstruktur dari sebuah gambar

tanpa mengandalkan ekstraksi teks biasa. Pendekatan ini memastikan bahwa output mengikuti format yang konsisten dan dapat dengan mudah diurai dan diproses oleh komponen hilir aplikasi.

Kemampuan untuk memaksa pemanggilan fungsi dan menentukan struktur output yang diharapkan adalah fitur yang kuat dari penggunaan alat dalam aplikasi berbasis AI. Ini memungkinkan pengembang untuk memiliki kendali lebih besar atas output yang dihasilkan dan menyederhanakan integrasi data yang dihasilkan AI ke dalam alur kerja aplikasi mereka.

Eksekusi Fungsi

Anda telah mendefinisikan fungsi-fungsi, dan meminta AI Anda, yang memutuskan bahwa ia harus memanggil salah satu dari fungsi Anda. Sekarang saatnya kode aplikasi Anda atau pustaka, jika Anda menggunakan gem Ruby seperti `raix-rails` untuk mengirimkan pemanggilan fungsi dan parameter-parameternya ke implementasi yang sesuai *dalam kode aplikasi Anda*.

Kode aplikasi Anda yang menentukan apa yang harus dilakukan dengan hasil eksekusi fungsi tersebut. Mungkin yang perlu dilakukan hanya melibatkan satu baris kode dalam lambda, atau mungkin melibatkan pemanggilan API eksternal. Mungkin melibatkan pemanggilan komponen AI lain, atau mungkin melibatkan ratusan atau bahkan ribuan baris kode dalam sistem Anda yang lain. Semuanya terserah Anda.

Terkadang pemanggilan fungsi adalah akhir dari operasi, tetapi jika hasilnya merepresentasikan informasi dalam yang akan dilanjutkan oleh AI, maka kode aplikasi Anda perlu memasukkan hasil eksekusi ke dalam transkrip obrolan dan membiarkan AI melanjutkan pemrosesannya.

Sebagai contoh, berikut adalah deklarasi fungsi `Raix` yang digunakan oleh `AccountManager Olympia` untuk berkomunikasi dengan klien kami sebagai bagian dari Orkestrasi Alur Kerja Cerdas untuk layanan pelanggan.

```
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
```

Mungkin tidak langsung jelas apa yang terjadi di sini, jadi saya akan menjelaskannya.

1. Kelas `AccountManager` mendefinisikan banyak fungsi yang terkait dengan pengelolaan akun. Kelas ini dapat mengubah paket langganan Anda, menambah dan menghapus anggota tim, dan hal-hal lainnya.
2. Instruksi tingkat atasnya memberi tahu `AccountManager` bahwa ia harus memberi tahu pemilik akun tentang hasil permintaan perubahan akun, menggunakan fungsi `notify_account_owner`.
3. Definisi ringkas dari fungsi tersebut mencakup:
 - nama
 - deskripsi
 - parameter `message: { type: "string" }`
 - blok yang akan dieksekusi ketika fungsi dipanggil

Setelah memperbarui transkrip dengan hasil dari blok fungsi, metode `chat_completion` dipanggil kembali. Metode ini bertanggung jawab untuk mengirimkan transkrip percakapan yang telah diperbarui kembali ke model AI untuk pemrosesan lebih lanjut. Kita menyebut proses ini sebagai *siklus percakapan*.

Ketika model AI menerima permintaan chat completion baru dengan transkrip yang diperbarui, ia memiliki akses ke hasil dari fungsi yang sebelumnya dieksekusi. Model dapat menganalisis hasil ini, memasukkannya ke dalam proses pengambilan keputusan, dan menghasilkan respons atau tindakan berikutnya berdasarkan konteks kumulatif dari percakapan. Model dapat memilih untuk mengeksekusi fungsi tambahan berdasarkan konteks yang diperbarui, atau dapat menghasilkan respons akhir untuk prompt asli jika menentukan bahwa tidak diperlukan pemanggilan fungsi lebih lanjut.

Kelanjutan Opsional dari Prompt Asli

Ketika Anda mengirimkan hasil tool kembali ke LLM dan melanjutkan pemrosesan prompt asli, AI menggunakan hasil tersebut untuk memanggil fungsi tambahan atau menghasilkan respons teks biasa final.



Beberapa model seperti [Command-R](#) dari Cohere dapat mencantumkan tool spesifik yang mereka gunakan dalam respons mereka, memberikan transparansi dan kemampuan pelacakan tambahan.

Tergantung pada model yang digunakan, hasil dari pemanggilan fungsi akan berada dalam pesan transkrip yang memiliki peran khusus mereka sendiri atau tercermin dalam sintaks lain. Tetapi bagian pentingnya adalah data tersebut harus ada dalam transkrip, sehingga dapat dipertimbangkan oleh AI saat memutuskan apa yang harus dilakukan selanjutnya.



Kesalahan umum (dan berpotensi mahal) adalah lupa menambahkan hasil fungsi ke transkrip sebelum melanjutkan chat. Akibatnya, AI akan mendapat prompt dengan cara yang pada dasarnya sama seperti sebelum ia memanggil fungsi pertama kali. Dengan kata lain, sejauh yang AI ketahui, ia belum memanggil fungsi tersebut. Jadi ia memanggilmnya lagi. Dan lagi. Dan lagi, selamanya sampai Anda menghentikannya. Semoga konteks Anda tidak terlalu besar, dan model Anda tidak terlalu mahal!

Praktik Terbaik untuk Penggunaan Tool

Untuk memaksimalkan penggunaan tool, pertimbangkan praktik-praktik terbaik berikut.

Definisi Deskriptif

Berikan nama dan deskripsi yang jelas dan deskriptif untuk setiap tool dan parameter masukannya. Ini membantu LLM lebih memahami tujuan dan kemampuan setiap tool.

Dari pengalaman saya, kata-kata bijak yang mengatakan bahwa “penamaan itu sulit” berlaku di sini; saya telah melihat hasil yang sangat berbeda dari LLM hanya dengan mengubah nama fungsi atau kata-kata dalam deskripsi. Terkadang menghapus deskripsi justru *meningkatkan* performa.

Pemrosesan Hasil Tool

Ketika meneruskan hasil tool kembali ke LLM, pastikan hasilnya terstruktur dengan baik dan komprehensif. Gunakan kunci dan nilai yang bermakna untuk merepresentasikan keluaran dari setiap tool. Lakukan eksperimen dengan format yang berbeda dan lihat mana yang bekerja paling baik, mulai dari JSON hingga teks biasa.

[Result Interpreter](#) mengatasi tantangan ini dengan menggunakan AI untuk menganalisis hasil dan memberikan penjelasan yang mudah dipahami manusia, ringkasan, atau poin-poin penting.

Penanganan Error

Terapkan mekanisme penanganan error yang kuat untuk menangani kasus di mana LLM mungkin menghasilkan parameter masukan yang tidak valid atau tidak didukung untuk pemanggilan tool. Tangani dan pulihkan dengan baik dari setiap error yang mungkin terjadi selama eksekusi tool.

Salah satu kualitas yang sangat baik dari AI adalah bahwa ia memahami pesan error! Yang berarti jika Anda bekerja dengan pola pikir cepat dan praktis, Anda bisa menangkap exception apa pun yang dihasilkan dalam implementasi sebuah tool, dan meneruskannya kembali ke AI sehingga AI tahu apa yang terjadi!

Sebagai contoh, berikut adalah versi yang disederhanakan dari implementasi pencarian google di 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
```

Pencarian Google di Olympia adalah proses dua langkah. Pertama Anda melakukan pencarian, kemudian Anda meringkas hasilnya. Jika terjadi kegagalan, apa pun itu, pesan kesalahan dikemas dan dikirim kembali ke AI. Teknik ini adalah dasar dari hampir semua pola *Penanganan Kesalahan Cerdas*

Misalnya, katakanlah panggilan API GoogleSearch gagal karena pengecualian 503 Service Unavailable. Hal ini mengalir ke atas ke penanganan kesalahan tingkat atas, dan deskripsi kesalahan dikirim kembali ke AI sebagai hasil dari pemanggilan fungsi.

Alih-alih hanya memberikan layar kosong atau kesalahan teknis kepada pengguna, AI mengatakan sesuatu seperti “Maaf, saya tidak dapat mengakses kemampuan Pencarian Google saya saat ini. Saya bisa mencoba lagi nanti, jika Anda mau.”

Ini mungkin terlihat hanya seperti trik cerdas, tetapi pertimbangkan jenis kesalahan yang berbeda, di mana AI memanggil API eksternal dan memiliki kontrol langsung atas parameter yang akan diteruskan ke API tersebut. Mungkin AI membuat kesalahan dalam cara menghasilkan parameter tersebut? Asalkan pesan kesalahan dari API eksternal cukup detail, meneruskan pesan kesalahan kembali ke AI yang memanggil berarti AI dapat mempertimbangkan kembali parameter tersebut dan mencoba lagi. Secara otomatis. Tidak peduli apa pun kesalahannya.

Sekarang pikirkan apa yang diperlukan untuk mereplikasi jenis penanganan kesalahan yang tangguh seperti itu dalam kode *normal*. Ini praktis tidak mungkin.

Penyempurnaan Iteratif

Jika LLM tidak merekomendasikan alat yang tepat atau menghasilkan respons yang kurang optimal, lakukan iterasi pada definisi alat, deskripsi, dan parameter masukan. Terus sempurnakan dan tingkatkan pengaturan alat berdasarkan perilaku yang diamati dan hasil yang diinginkan.

1. Mulai dengan definisi alat sederhana: Mulailah dengan mendefinisikan alat dengan nama, deskripsi, dan parameter masukan yang jelas dan ringkas. Hindari membuat pengaturan alat yang terlalu rumit pada awalnya dan fokus pada fungsi intinya. Misalnya, jika Anda ingin menyimpan hasil analisis sentimen, mulailah dengan definisi dasar seperti:

```

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. Uji dan amati: Setelah definisi alat awal terpasang, ujilah dengan berbagai prompt dan amati bagaimana LLM berinteraksi dengan alat tersebut. Perhatikan kualitas dan relevansi dari respons yang dihasilkan. Jika LLM menghasilkan respons yang suboptimal, saatnya untuk menyempurnakan definisi alat.
3. Sempurnakan deskripsi: Jika LLM salah memahami tujuan suatu alat, cobalah menyempurnakan deskripsi alat tersebut. Berikan lebih banyak konteks, contoh, atau klarifikasi untuk memandu LLM dalam menggunakan alat secara efektif. Misalnya, Anda dapat memperbarui deskripsi alat analisis sentimen untuk lebih spesifik menangani *nada emosional* dari teks yang sedang dianalisis:

```

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. Sesuaikan parameter masukan: Jika LLM menghasilkan parameter masukan yang tidak valid atau tidak relevan untuk suatu alat, pertimbangkan untuk menyesuaikan definisi parameter. Tambahkan batasan yang lebih spesifik, aturan validasi, atau contoh untuk memperjelas format masukan yang diharapkan.

5. Lakukan iterasi berdasarkan umpan balik: Pantau terus kinerja alat Anda dan kumpulkan umpan balik dari pengguna atau pemangku kepentingan. Gunakan umpan balik ini untuk mengidentifikasi area yang perlu ditingkatkan dan lakukan penyempurnaan iteratif pada definisi alat. Sebagai contoh, jika pengguna melaporkan bahwa analisis tidak menangani sarkasme dengan baik, Anda dapat menambahkan catatan dalam deskripsi:

```
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 }
```

Dengan menyempurnakan definisi alat Anda secara iteratif berdasarkan perilaku yang diamati dan umpan balik, Anda dapat secara bertahap meningkatkan kinerja dan efektivitas aplikasi berbasis AI Anda. Ingatlah untuk menjaga agar definisi alat tetap jelas, ringkas, dan terfokus pada tugas spesifik yang sedang ditangani. Lakukan pengujian dan validasi interaksi alat secara rutin untuk memastikan semuanya selaras dengan hasil yang Anda inginkan.

Menyusun dan Merantai Alat

Salah satu aspek paling kuat dari penggunaan alat yang baru disinggung sejauh ini adalah kemampuan untuk menyusun dan merantai beberapa alat bersama-sama untuk menyelesaikan tugas-tugas kompleks. Dengan merancang definisi alat dan format masukan/keluarannya secara cermat, Anda dapat membuat komponen yang dapat digunakan ulang dan dikombinasikan dengan berbagai cara.

Mari kita lihat contoh di mana Anda sedang membangun jalur analisis data untuk aplikasi berbasis AI Anda. Anda mungkin memiliki alat-alat berikut:

1. **DataRetrieval**: Sebuah alat yang mengambil data dari basis data atau API berdasarkan kriteria tertentu.
2. **DataProcessing**: Sebuah alat yang melakukan perhitungan, transformasi, atau agregasi pada data yang diambil.
3. **DataVisualization**: Sebuah alat yang menyajikan data yang telah diproses dalam format yang mudah dipahami pengguna, seperti grafik atau bagan.

Dengan merantai alat-alat ini bersama-sama, Anda dapat membuat alur kerja yang kuat yang mengambil data yang relevan, memprosesnya, dan menyajikan hasilnya dengan cara yang bermakna. Berikut adalah bagaimana alur kerja penggunaan alat mungkin terlihat:

1. LLM menerima permintaan pengguna yang meminta wawasan tentang data penjualan untuk kategori produk tertentu.
2. LLM memilih alat `DataRetrieval` dan menghasilkan parameter masukan yang sesuai untuk mengambil data penjualan yang relevan dari basis data.
3. Data yang diambil “diteruskan” ke alat `DataProcessing`, yang menghitung metrik seperti total pendapatan, harga penjualan rata-rata, dan tingkat pertumbuhan.
4. Data yang telah diproses kemudian diolah oleh alat `DataVisualization`, yang membuat grafik atau bagan yang menarik secara visual untuk merepresentasikan wawasan tersebut, meneruskan URL grafik kembali ke LLM.
5. Akhirnya, LLM menghasilkan respons terformat untuk permintaan pengguna menggunakan markdown, menggabungkan data yang telah divisualisasikan dan memberikan ringkasan temuan utama.

Dengan menyusun alat-alat ini bersama-sama, Anda dapat membuat alur kerja analisis data yang mulus yang dapat dengan mudah diintegrasikan ke dalam aplikasi Anda. Keindahan pendekatan ini adalah bahwa setiap alat dapat dikembangkan dan diuji

secara independen, dan kemudian dikombinasikan dengan cara yang berbeda untuk menyelesaikan berbagai masalah.

Untuk memungkinkan komposisi dan perangkaian alat yang lancar, penting untuk mendefinisikan format masukan dan keluaran yang jelas untuk setiap alat.

Sebagai contoh, alat `DataRetrieval` mungkin menerima parameter seperti detail koneksi database, nama tabel, dan kondisi query, dan mengembalikan hasil dalam bentuk objek JSON terstruktur. Alat `DataProcessing` kemudian dapat menerima objek JSON ini sebagai masukan dan menghasilkan objek JSON yang telah ditransformasi sebagai keluaran. Dengan menstandarkan aliran data antar alat, Anda dapat memastikan kompatibilitas dan kemampuan penggunaan ulang.

Saat Anda merancang ekosistem alat Anda, pikirkan bagaimana alat-alat yang berbeda dapat dikombinasikan untuk menangani kasus penggunaan umum dalam aplikasi Anda. Pertimbangkan untuk membuat alat tingkat tinggi yang mengenkapsulasi alur kerja atau logika bisnis yang umum, sehingga memudahkan LLM untuk memilih dan menggunakannya secara efektif.

Ingat, kekuatan penggunaan alat terletak pada fleksibilitas dan modularitas yang diberikannya. Dengan memecah tugas-tugas kompleks menjadi alat-alat yang lebih kecil dan dapat digunakan kembali, Anda dapat menciptakan aplikasi berbasis AI yang tangguh dan adaptif yang dapat mengatasi berbagai tantangan.

Arah Masa Depan

Seiring berkembangnya bidang pengembangan aplikasi berbasis AI, kita dapat mengharapkan kemajuan lebih lanjut dalam kemampuan penggunaan alat. Beberapa arah masa depan yang potensial meliputi:

1. **Penggunaan Alat Multi-Langkah:** LLM mungkin dapat memutuskan berapa kali mereka perlu menggunakan alat untuk menghasilkan respons yang memuaskan.

Ini bisa melibatkan beberapa putaran pemilihan dan eksekusi alat berdasarkan hasil antara.

2. **Alat yang Telah Ditentukan:** Platform AI mungkin menyediakan seperangkat alat yang telah ditentukan yang dapat dimanfaatkan pengembang secara langsung, seperti interpreter Python, alat pencarian web, atau fungsi utilitas umum.
3. **Integrasi Mulus:** Seiring penggunaan alat menjadi lebih umum, kita dapat mengharapkan integrasi yang lebih baik antara platform AI dan kerangka kerja pengembangan populer, memudahkan pengembang untuk memasukkan penggunaan alat ke dalam aplikasi mereka.

Penggunaan alat adalah teknik yang kuat yang memungkinkan pengembang untuk memanfaatkan potensi penuh LLM dalam aplikasi berbasis AI. Dengan menghubungkan LLM ke alat dan sumber daya eksternal, Anda dapat menciptakan sistem yang lebih dinamis, cerdas, dan peka konteks yang dapat beradaptasi dengan kebutuhan pengguna dan memberikan wawasan serta tindakan yang berharga.

Meskipun penggunaan alat menawarkan kemungkinan yang sangat besar, penting untuk menyadari potensi tantangan dan pertimbangan yang ada. Salah satu aspek kunci adalah mengelola kompleksitas interaksi alat dan memastikan stabilitas serta keandalan sistem secara keseluruhan. Anda perlu menangani skenario di mana pemanggilan alat mungkin gagal, mengembalikan hasil yang tidak terduga, atau memiliki implikasi kinerja. Selain itu, Anda harus mempertimbangkan langkah-langkah keamanan dan kontrol akses untuk mencegah penggunaan alat yang tidak sah atau berbahaya. Mekanisme penanganan kesalahan, pencatatan log, dan pemantauan yang tepat sangat penting untuk menjaga integritas dan kinerja aplikasi berbasis AI Anda.

Saat Anda mengeksplorasi berbagai kemungkinan penggunaan alat dalam proyek Anda sendiri, ingatlah untuk memulai dengan tujuan yang jelas, merancang definisi alat yang terstruktur dengan baik, dan melakukan iterasi berdasarkan umpan balik dan hasil.

Dengan pendekatan dan pola pikir yang tepat, penggunaan alat dapat membuka tingkat inovasi dan nilai baru dalam aplikasi berbasis AI Anda

Pemrosesan Stream



Pengaliran data melalui HTTP, yang juga dikenal sebagai server-sent events (SSE), adalah mekanisme di mana server terus mengirimkan data ke klien saat data tersedia, tanpa perlu klien memintanya secara eksplisit. Karena respons AI dihasilkan secara bertahap, masuk akal untuk memberikan pengalaman pengguna yang responsif dengan menampilkan keluaran AI saat sedang dihasilkan. Dan sebenarnya semua API penyedia AI yang saya ketahui menawarkan respons streaming sebagai pilihan dalam endpoint penyelesaian mereka.

Alasan mengapa bab ini muncul di sini dalam buku ini, tepat setelah [Menggunakan Alat](#) adalah karena betapa kuatnya menggabungkan penggunaan alat dengan respons AI langsung kepada pengguna. Hal ini memungkinkan pengalaman yang dinamis dan interaktif di mana AI dapat memproses masukan pengguna, memanfaatkan berbagai alat dan fungsi sesuai kebutuhannya, dan kemudian memberikan respons secara real-time.

Untuk mencapai interaksi yang mulus ini, Anda perlu menulis penanganan stream yang dapat mengirimkan pemanggilan fungsi alat yang dipanggil AI serta keluaran teks biasa kepada pengguna akhir. Kebutuhan untuk melakukan perulangan setelah memproses fungsi alat menambahkan tantangan menarik pada pekerjaan ini.

Mengimplementasikan ReplyStream

Untuk menunjukkan bagaimana pemrosesan stream dapat diimplementasikan, bab ini akan membahas secara mendalam versi yang disederhanakan dari kelas ReplyStream yang digunakan di Olympia. Instance dari kelas ini dapat dilewatkan sebagai parameter stream dalam pustaka klien AI seperti [ruby-openai](#) dan [openrouter](#)

Berikut cara saya menggunakan ReplyStream di PromptSubscriber Olympia, yang mendengarkan melalui Wisper untuk pembuatan pesan pengguna baru.

```
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...
```

Selain memiliki referensi context ke pelanggan prompt yang membuat instansnya, kelas ReplyStream juga memiliki variabel instans untuk menyimpan penyangga data yang diterima, dan array untuk melacak nama-nama fungsi dan argumen yang dipanggil selama pemrosesan aliran.

```
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
```

Metode `initialize` menyiapkan keadaan awal dari instansi `ReplyStream`, menginisialisasi buffer, konteks, dan variabel-variabel lainnya.

Metode `call` adalah titik masuk utama untuk memproses data streaming. Metode ini menerima sebuah chunk data (yang direpresentasikan sebagai hash) dan parameter opsional `bytesize`, yang dalam contoh kita tidak digunakan. Di dalam metode ini, kelas menggunakan pencocokan pola untuk menangani berbagai skenario berdasarkan struktur potongan data yang diterima.



Memanggil `deep_symbolize_keys` pada chunk membantu membuat pencocokan pola lebih elegan, dengan memungkinkan kita mengoperasikan simbol daripada string.

```
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
```

Pola pertama yang kita cocokkan adalah pemanggilan alat beserta nama fungsinya. Jika kita mendeteksi satu, kita memasukkannya ke dalam larik `f_name`. Kita menyimpan nama-nama fungsi dalam larik terindeks, karena model mampu melakukan pemanggilan fungsi secara paralel, mengirimkan lebih dari satu fungsi untuk dieksekusi pada waktu yang sama.

Pemanggilan fungsi paralel adalah kemampuan model AI untuk melakukan beberapa pemanggilan fungsi secara bersamaan, memungkinkan efek dan hasil dari pemanggilan fungsi ini diselesaikan secara paralel. Ini sangat berguna jika fungsi membutuhkan waktu lama, dan mengurangi komunikasi bolak-balik dengan API, yang pada gilirannya dapat menghemat sejumlah besar pengeluaran token.

Selanjutnya kita perlu mencocokkan argumen yang sesuai dengan pemanggilan fungsi tersebut.

```

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

```

Serupa dengan cara kita menangani nama fungsi, kita menyimpan argumen dalam array terindeks.

Selanjutnya, kita mencari pesan yang ditampilkan ke pengguna, yang akan tiba dari server satu token pada satu waktu dan akan ditetapkan ke variabel `new_content`. Kita juga perlu memperhatikan `finish_reason`. Nilainya akan tetap `nil` sampai potongan terakhir dari urutan keluaran.

```

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

```

Yang penting, kita menambahkan ekspresi pencocokan pola untuk menangani pesan kesalahan yang dikirim oleh penyedia model AI. Dalam lingkungan pengembangan

lokal, kita memunculkan eksepsi, tetapi dalam produksi, kita mencatat kesalahan tersebut dan menyelesaikan prosesnya.

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

Klausa else terakhir dari case akan dijalankan jika tidak ada pola sebelumnya yang cocok. Ini hanyalah tindakan pengamanan sehingga jika model AI mulai mengirimkan potongan yang tidak dikenali kepada kita, kita akan mengetahuinya.

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

Metode `send_to_client` bertanggung jawab untuk mengirimkan konten yang terbuffer kepada klien. Metode ini memeriksa bahwa buffer tidak kosong, memperbarui konten pesan bot, merender pesan bot, dan menyimpan konten tersebut dalam basis data untuk memastikan persistensi data.

```
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
```

Metode `finalize` dipanggil ketika pemrosesan stream selesai. Metode ini mengirimkan pemanggilan fungsi jika ada yang diterima selama stream berlangsung, memperbarui pesan bot dengan konten final dan informasi relevan lainnya, serta mengatur ulang riwayat pemanggilan fungsi

```
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
```

Jika model memutuskan untuk memanggil sebuah fungsi, Anda perlu “mengirim” pemanggilan fungsi tersebut (nama dan argumen) sedemikian rupa sehingga fungsi

tersebut dapat dieksekusi dan pesan `function_call` serta `function_result` ditambahkan ke dalam transkrip percakapan

Berdasarkan pengalaman saya, lebih baik menangani pembuatan pesan-pesan fungsi di satu tempat dalam basis kode Anda, daripada mengandalkan implementasi alat. Ini bukan hanya membuat kode lebih rapi, tapi juga memiliki alasan praktis yang sangat penting: jika model AI memanggil sebuah fungsi, dan tidak melihat pesan pemanggilan dan hasil dalam transkrip ketika Anda melakukan perulangan, *model akan memanggil fungsi yang sama lagi*. Berpotensi terus-menerus. Ingat bahwa AI bersifat sepenuhnya stateless, jadi kecuali Anda mengembalikan pemanggilan fungsi tersebut kepadanya, bagi AI pemanggilan tersebut tidak pernah terjadi.

```
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
```



Membersihkan riwayat pemanggilan fungsi setelah pengiriman sama pentingnya dengan memastikan pemanggilan dan hasilnya masuk ke dalam transkrip Anda, sehingga Anda tidak terus-menerus memanggil fungsi yang sama berulang-ulang setiap kali Anda melakukan perulangan.

“Putaran Percakapan”

Saya terus menyebutkan perulangan, tetapi jika Anda baru mengenal pemanggilan fungsi, mungkin tidak jelas *mengapa* kita perlu melakukan perulangan. Alasannya

adalah ketika AI “meminta” Anda untuk mengeksekusi fungsi alat atas namanya, ia akan berhenti merespons. Anda yang bertanggung jawab untuk mengeksekusi fungsi-fungsi tersebut, mengumpulkan hasilnya, menambahkan hasil ke dalam transkrip, dan kemudian mengirimkan prompt awal lagi untuk mendapatkan sekumpulan pemanggilan fungsi baru atau hasil yang ditujukan untuk pengguna.

Dalam kelas `PromptSubscriber`, kita menggunakan metode `prompt` dari modul `PromptDeclarations` untuk mendefinisikan perilaku putaran percakapan. Parameter `until` diatur ke `-> { bot_message.complete? }`, yang berarti perulangan akan berlanjut sampai `bot_message` ditandai sebagai selesai.

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



Tetapi kapan `bot_message` ditandai sebagai selesai? Jika Anda lupa, lihat kembali baris 13 dari metode `finalize`.

Mari kita tinjau seluruh logika pemrosesan aliran.

1. `PromptSubscriber` menerima pesan pengguna baru melalui metode `message_created`, yang dipanggil oleh sistem pub/sub Wisper setiap kali pengguna akhir membuat prompt baru.
2. Metode kelas `prompt` secara deklaratif mendefinisikan perilaku logika penyelesaian chat untuk `PromptSubscriber`. Model AI akan menjalankan penyelesaian chat dengan konten pesan pengguna, instance baru `ReplyStream` sebagai parameter aliran, dan kondisi perulangan yang ditentukan.
3. Model AI memproses prompt dan mulai menghasilkan respons. Saat respons dialirkan, metode `call` dari instance `ReplyStream` dipanggil untuk setiap bagian data.

4. Jika model AI memutuskan untuk memanggil fungsi alat, nama fungsi dan argumen diekstrak dari bagian tersebut dan disimpan masing-masing dalam array `f_name` dan `f_arguments`.
5. Jika model AI menghasilkan konten yang ditampilkan ke pengguna, konten tersebut di-buffer dan dikirim ke klien melalui metode `send_to_client`.
6. Setelah pemrosesan aliran selesai, metode `finalize` dipanggil. Jika ada fungsi alat yang dipanggil selama aliran, fungsi-fungsi tersebut dikirim menggunakan metode `dispatch` dari `PromptSubscriber`.
7. Metode `dispatch` menambahkan pesan `function_call` ke transkrip percakapan, menjalankan fungsi alat yang sesuai, dan menambahkan pesan `function_result` ke transkrip dengan hasil dari pemanggilan fungsi.
8. Setelah mengirim fungsi-fungsi alat, riwayat pemanggilan fungsi dihapus untuk mencegah pemanggilan fungsi ganda dalam perulangan berikutnya.
9. Jika tidak ada fungsi alat yang dipanggil, metode `finalize` memperbarui `bot_message` dengan konten final, menandainya sebagai selesai, dan mengirim pesan yang diperbarui ke klien.
10. Kondisi perulangan `-> { bot_message.complete? }` dievaluasi. Jika `bot_message` tidak ditandai sebagai selesai, perulangan berlanjut, dan prompt asli dikirim kembali dengan transkrip percakapan yang diperbarui.
11. Langkah 3-10 diulang sampai `bot_message` ditandai sebagai selesai, yang menunjukkan bahwa model AI telah selesai menghasilkan responnya dan tidak ada lagi fungsi alat yang perlu dijalankan.

Dengan mengimplementasikan putaran percakapan ini, Anda memungkinkan model AI untuk terlibat dalam interaksi bolak-balik dengan aplikasi, menjalankan fungsi-fungsi alat sesuai kebutuhan dan menghasilkan respons yang ditujukan untuk pengguna sampai percakapan mencapai kesimpulan yang alami.

Kombinasi pemrosesan aliran dan putaran percakapan memungkinkan pengalaman berbasis AI yang dinamis dan interaktif, di mana model AI dapat memproses masukan

pengguna, memanfaatkan berbagai alat dan fungsi, serta memberikan respons secara real-time berdasarkan konteks percakapan yang berkembang.

Kelanjutan Otomatis

Penting untuk menyadari keterbatasan output AI. Kebanyakan model memiliki jumlah maksimum token yang dapat mereka hasilkan dalam satu respons, yang ditentukan oleh parameter `max_tokens`. Jika model AI mencapai batas ini saat menghasilkan respons, ia akan berhenti secara tiba-tiba dan menunjukkan bahwa output telah terpotong.

Dalam respons streaming dari API platform AI, Anda dapat mendeteksi situasi ini dengan memeriksa variabel `finish_reason` dalam chunk. Jika `finish_reason` diatur ke "length" (atau beberapa nilai kunci lain yang spesifik untuk model tersebut), ini berarti bahwa model telah mencapai batas token maksimumnya selama proses generasi dan output telah terpotong.

Salah satu cara untuk menangani skenario ini dengan baik dan memberikan pengalaman pengguna yang mulus adalah dengan mengimplementasikan mekanisme kelanjutan otomatis dalam logika pemrosesan aliran Anda. Dengan menambahkan pencocokan pola untuk alasan penyelesaian terkait panjang, Anda dapat memilih untuk mengulang dan secara otomatis melanjutkan output dari tempat terakhir ia berhenti.

Berikut adalah contoh yang sengaja disederhanakan tentang bagaimana Anda dapat memodifikasi metode `call` dalam kelas `ReplyStream` untuk mendukung kelanjutan otomatis:

```
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
```

Dalam versi yang dimodifikasi ini, ketika `finish_reason` menunjukkan output yang terpotong, alih-alih menyelesaikan aliran, kita menambahkan sepasang pesan ke transkrip tanpa menyelesaikannya, memindahkan pesan respons yang ditujukan untuk pengguna ke “bawah” transkrip dengan memperbarui atribut `created_at`-nya, dan kemudian membiarkan loop terjadi, sehingga AI dapat melanjutkan dari tempat terakhir ia berhenti.

Ingat bahwa endpoint penyelesaian AI bersifat stateless. Ia hanya “mengetahui” apa yang Anda beritahukan melalui transkrip. Dalam kasus ini, cara kita memberi tahu AI bahwa responsnya terpotong adalah dengan menambahkan pesan “tak terlihat” (bagi pengguna akhir) ke dalam transkrip. Namun ingat, ini adalah contoh yang sengaja disederhanakan. Implementasi yang sebenarnya akan memerlukan pengelolaan transkrip lebih lanjut untuk memastikan bahwa kita tidak membuang token dan/atau membingungkan AI dengan pesan asisten yang terduplikasi dalam transkrip.

Implementasi auto-continuation yang sebenarnya juga harus memiliki apa yang disebut “logika pemutus sirkuit” untuk mencegah perulangan yang tak terkendali. Alasannya adalah, dengan jenis prompt pengguna tertentu dan pengaturan `max_tokens` yang rendah, AI bisa terus mengulang output yang ditujukan untuk pengguna tanpa henti.

Perlu diingat bahwa setiap perulangan memerlukan permintaan terpisah, dan setiap permintaan mengkonsumsi seluruh transkrip Anda lagi. Anda harus mempertimbangkan dengan baik trade-off antara pengalaman pengguna dan penggunaan API ketika memutuskan apakah akan mengimplementasikan auto-continuation dalam aplikasi Anda. Auto-continuation secara khusus bisa menjadi sangat mahal, terutama ketika menggunakan model komersial premium.

Kesimpulan

Pemrosesan aliran adalah aspek penting dalam membangun aplikasi berbasis AI yang menggabungkan penggunaan alat dengan respons AI secara langsung. Dengan menangani data streaming dari API platform AI secara efisien, Anda dapat memberikan pengalaman pengguna yang mulus dan interaktif, menangani respons besar, mengoptimalkan penggunaan sumber daya, dan menangani kesalahan dengan baik.

Kelas `Conversation::ReplyStream` yang disediakan mendemonstrasikan

bagaimana pemrosesan aliran dapat diimplementasikan dalam aplikasi Ruby menggunakan pencocokan pola dan arsitektur berbasis event. Dengan memahami dan memanfaatkan teknik pemrosesan aliran, Anda dapat membuka potensi penuh integrasi AI dalam aplikasi Anda dan memberikan pengalaman pengguna yang kuat dan menarik.

Data yang Dapat Memperbaiki Diri



Data yang dapat memperbaiki diri adalah pendekatan yang ampuh untuk memastikan integritas, konsistensi, dan kualitas data dalam aplikasi dengan memanfaatkan kemampuan model bahasa besar (LLM). Kategori pola ini berfokus pada gagasan penggunaan AI untuk secara otomatis mendeteksi, mendiagnosis, dan memperbaiki anomali, inkonsistensi, atau kesalahan data, sehingga mengurangi beban pengembang dan mempertahankan tingkat keandalan data yang tinggi.

Pada intinya, pola data yang dapat memperbaiki diri mengakui bahwa data adalah urat nadi dari setiap aplikasi, dan memastikan akurasi serta integritasnya sangat penting untuk fungsi dan pengalaman pengguna aplikasi yang tepat. Namun, mengelola dan mempertahankan kualitas data bisa menjadi tugas yang kompleks dan memakan waktu,

terutama ketika aplikasi berkembang dalam ukuran dan kompleksitas. Di sinilah kekuatan AI berperan.

Dalam pola data yang dapat memperbaiki diri, pekerja AI digunakan untuk terus memantau dan menganalisis data aplikasi Anda. Model-model ini memiliki kemampuan untuk memahami dan menginterpretasikan pola, hubungan, dan anomali dalam data. Dengan memanfaatkan kemampuan pemrosesan dan pemahaman bahasa alami mereka, mereka dapat mengidentifikasi potensi masalah atau inkonsistensi dalam data dan mengambil tindakan yang tepat untuk memperbaikinya.

Proses data yang dapat memperbaiki diri biasanya melibatkan beberapa langkah kunci:

1. **Pemantauan Data:** Pekerja AI terus memantau aliran data, basis data, atau sistem penyimpanan aplikasi, mencari tanda-tanda anomali, inkonsistensi, atau kesalahan. Atau, Anda dapat mengaktifkan komponen AI sebagai reaksi terhadap pengecualian.
2. **Deteksi Anomali:** Ketika masalah terdeteksi, pekerja AI menganalisis data secara detail untuk mengidentifikasi sifat dan cakupan spesifik dari masalah tersebut. Ini bisa melibatkan pendeteksian nilai yang hilang, format yang tidak konsisten, atau data yang melanggar aturan atau batasan yang telah ditentukan.
3. **Diagnosis dan Koreksi:** Setelah masalah teridentifikasi, pekerja AI menggunakan pengetahuan dan pemahamannya tentang domain data untuk menentukan tindakan yang tepat. Ini bisa melibatkan perbaikan data secara otomatis, mengisi nilai yang hilang, atau menandai masalah untuk intervensi manusia jika diperlukan.
4. **Pembelajaran Berkelanjutan (opsional, tergantung kasus penggunaan):** Ketika pekerja AI Anda menemui dan menyelesaikan berbagai masalah data, ia dapat menghasilkan output yang menjelaskan apa yang terjadi dan bagaimana ia meresponnya. Metadata ini dapat dimasukkan ke dalam proses pembelajaran yang memungkinkan Anda (dan mungkin model yang mendasarinya, melalui

penyetelan halus) untuk menjadi lebih efektif dan efisien dari waktu ke waktu dalam mengidentifikasi dan menyelesaikan anomali data.

Dengan mendeteksi dan memperbaiki masalah data secara otomatis, Anda dapat memastikan aplikasi Anda beroperasi dengan data yang berkualitas tinggi dan dapat diandalkan. Hal ini mengurangi risiko kesalahan, inkonsistensi, atau bug terkait data yang dapat memengaruhi fungsionalitas atau pengalaman pengguna aplikasi.

Setelah Anda memiliki pekerja AI yang menangani tugas pemantauan dan perbaikan data, Anda dapat memfokuskan upaya Anda pada aspek-aspek penting lainnya dari aplikasi. Ini menghemat waktu dan sumber daya yang seharusnya digunakan untuk pembersihan dan pemeliharaan data secara manual. Faktanya, seiring aplikasi Anda bertumbuh dalam ukuran dan kompleksitas, pengelolaan kualitas data secara manual menjadi semakin menantang. Pola “Data yang Dapat Memperbaiki Diri” dapat berskala secara efektif dengan memanfaatkan kekuatan AI untuk menangani volume data yang besar dan mendeteksi masalah secara real-time.



Karena sifatnya, model AI dapat beradaptasi dengan perubahan pola data, skema, atau persyaratan seiring waktu dengan sedikit atau tanpa pengawasan. Selama arahan mereka memberikan panduan yang memadai, terutama mengenai hasil yang diinginkan, aplikasi Anda mungkin dapat berkembang dan menangani skenario data baru tanpa memerlukan intervensi manual atau perubahan kode yang ekstensif.

Pola data yang dapat memperbaiki diri selaras dengan kategori pola lain yang telah kita bahas, seperti “Multitude of Workers”. Kemampuan data yang dapat memperbaiki diri dapat dipandang sebagai jenis pekerja khusus yang berfokus secara spesifik pada memastikan kualitas dan integritas data. Pekerja jenis ini beroperasi bersama pekerja AI lainnya, masing-masing berkontribusi pada aspek yang berbeda dari fungsionalitas aplikasi.

Implementasi pola data yang dapat memperbaiki diri dalam praktik memerlukan desain yang cermat dan integrasi model AI ke dalam arsitektur aplikasi. Karena adanya risiko kehilangan dan kerusakan data, Anda harus menentukan panduan yang jelas tentang bagaimana Anda akan menggunakan teknik ini. Anda juga harus mempertimbangkan faktor-faktor seperti kinerja, skalabilitas, dan keamanan data.

Studi Kasus Praktis: Memperbaiki JSON yang Rusak

Salah satu cara paling praktis dan nyaman untuk memanfaatkan data yang dapat memperbaiki diri juga sangat sederhana untuk dijelaskan: memperbaiki JSON yang rusak.

Teknik ini dapat diterapkan pada tantangan umum dalam menangani data yang tidak sempurna atau tidak konsisten yang dihasilkan oleh LLM, seperti JSON yang rusak, dan menyediakan pendekatan untuk mendeteksi dan memperbaiki masalah ini secara otomatis.

Di Olympia, saya secara rutin menghadapi skenario di mana LLM menghasilkan data JSON yang tidak sepenuhnya valid. Ini dapat terjadi karena berbagai alasan, seperti LLM menambahkan komentar sebelum atau sesudah kode JSON yang sebenarnya, atau memperkenalkan kesalahan sintaks seperti koma yang hilang atau tanda kutip ganda yang tidak di-escape. Masalah-masalah ini dapat menyebabkan error parsing dan menimbulkan gangguan pada fungsionalitas aplikasi.

Untuk mengatasi masalah ini, saya telah mengimplementasikan sebuah solusi praktis dalam bentuk kelas `JsonFixer`. Kelas ini mengejawantahkan pola “Self-Healing Data” dengan mengambil JSON yang rusak sebagai input dan memanfaatkan LLM untuk memperbaikinya sambil mempertahankan sebanyak mungkin informasi dan maksud aslinya.

```
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" }
15
16     self.stop = ["```\n"]
17
18     chat_completion(json: true)
19   end
20
21   def model
22     "mistralai/mixtral-8x7b-instruct:nitro"
23   end
24 end
```



Perhatikan bagaimana `JsonFixer` menggunakan `Ventriloquist` untuk mengarahkan respons AI.

Proses penyembuhan mandiri data JSON bekerja sebagai berikut:

1. **Pembuatan JSON:** LLM digunakan untuk menghasilkan data JSON berdasarkan prompt atau persyaratan tertentu. Namun, karena sifat dasar LLM, JSON yang dihasilkan mungkin tidak selalu valid secara sempurna. Parser JSON tentu saja akan memunculkan `ParserError` jika Anda memberikannya JSON yang tidak valid.

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

Perhatikan bahwa pesan kesalahan juga dilewatkan ke pemanggilan `JSONFixer` sehingga tidak perlu sepenuhnya berasumsi apa yang salah dengan data tersebut, terutama karena parser sering kali akan memberi tahu Anda secara tepat apa yang salah.

2. **Koreksi Berbasis LLM:** Kelas `JSONFixer` mengirimkan JSON yang rusak kembali ke LLM, bersama dengan prompt atau instruksi khusus untuk memperbaiki JSON tersebut sambil mempertahankan informasi dan maksud asli sebanyak mungkin. LLM, yang dilatih dengan data dalam jumlah besar dan memiliki pemahaman tentang sintaks JSON, berusaha memperbaiki kesalahan dan menghasilkan string JSON yang valid. [Response Fencing](#) digunakan untuk membatasi output dari LLM, dan kami memilih Mixtral 8x7B sebagai model AI, karena sangat cocok untuk tugas semacam ini.
3. **Validasi dan Integrasi:** String JSON yang telah diperbaiki yang dikembalikan oleh LLM diparse oleh kelas `JSONFixer` itu sendiri, karena memanggil `chat_completion(json: true)`. Jika JSON yang diperbaiki lolos validasi, JSON tersebut diintegrasikan kembali ke dalam alur kerja aplikasi, memungkinkan aplikasi untuk melanjutkan pemrosesan data dengan lancar. JSON yang rusak telah “disembuhkan”.

Meskipun saya telah menulis dan menulis ulang implementasi `JSONFixer` saya sendiri beberapa kali, saya ragu bahwa total waktu yang diinvestasikan dalam semua versi tersebut lebih dari satu atau dua jam.

Perhatikan bahwa pelestarian maksud adalah elemen kunci dari setiap pola data penyembuhan mandiri. Proses koreksi berbasis LLM bertujuan untuk mempertahankan

informasi dan maksud asli dari JSON yang dihasilkan sebanyak mungkin. Ini memastikan bahwa JSON yang diperbaiki mempertahankan makna semantiknya dan dapat digunakan secara efektif dalam konteks aplikasi.

Implementasi praktis dari pendekatan “Data Penyembuhan Mandiri” di Olympia ini dengan jelas menunjukkan bagaimana AI, khususnya LLM, dapat dimanfaatkan untuk menyelesaikan tantangan data di dunia nyata. Ini menunjukkan kekuatan menggabungkan teknik pemrograman tradisional dengan kemampuan AI untuk membangun aplikasi yang kuat dan efisien.

Hukum Postel dan Pola “Data Penyembuhan Mandiri”

“Data Penyembuhan Mandiri,” seperti yang dicontohkan oleh kelas JSONFixer, sejalan dengan prinsip yang dikenal sebagai Hukum Postel, yang juga disebut sebagai Prinsip Ketangguhan. Hukum Postel menyatakan:

“Bersikaplah konservatif dalam apa yang Anda lakukan, bersikaplah liberal dalam apa yang Anda terima dari orang lain.”

Prinsip ini, yang awalnya diungkapkan oleh Jon Postel, seorang pelopor Internet awal, menekankan pentingnya membangun sistem yang toleran terhadap input yang beragam atau bahkan sedikit salah sambil mempertahankan kepatuhan yang ketat terhadap protokol yang ditentukan ketika mengirim output.

Dalam konteks “Self-Healing Data,” kelas JSONFixer mewujudkan Hukum Postel dengan bersikap toleran dalam menerima data JSON yang rusak atau tidak sempurna yang dihasilkan oleh LLM (Large Language Model). Kelas ini tidak langsung menolak atau gagal ketika menemui JSON yang tidak sepenuhnya mengikuti format yang diharapkan. Sebaliknya, ia mengambil pendekatan yang toleran dan berusaha memperbaiki JSON menggunakan kemampuan LLM.

Dengan bersikap toleran dalam menerima JSON yang tidak sempurna, kelas JSON-Fixer menunjukkan ketangguhan dan fleksibilitas. Kelas ini mengakui bahwa data di dunia nyata sering kali hadir dalam berbagai bentuk dan mungkin tidak selalu sesuai dengan spesifikasi yang ketat. Dengan menangani dan memperbaiki penyimpangan ini secara anggun, kelas ini memastikan bahwa aplikasi dapat terus berfungsi dengan lancar, bahkan saat menghadapi data yang tidak sempurna.

Di sisi lain, kelas JSONFixer juga mematuhi aspek konservatif dari Hukum Postel ketika berkaitan dengan output. Setelah memperbaiki JSON menggunakan LLM, kelas ini memvalidasi JSON yang telah diperbaiki untuk memastikan bahwa JSON tersebut benar-benar sesuai dengan format yang diharapkan. Ini menjaga integritas dan kebenaran data sebelum meneruskannya ke bagian lain dari aplikasi. Pendekatan konservatif ini menjamin bahwa output dari kelas JSONFixer dapat diandalkan dan konsisten, mendorong interoperabilitas dan mencegah penyebaran kesalahan.

Fakta Menarik tentang Jon Postel:

- Jon Postel (1943-1998) adalah ilmuwan komputer Amerika yang memainkan peran penting dalam pengembangan Internet. Dia dikenal sebagai “Dewa Internet” karena kontribusinya yang signifikan terhadap protokol dan standar yang mendasarinya.
- Postel adalah editor seri dokumen Request for Comments (RFC), yang merupakan serangkaian catatan teknis dan organisasional tentang Internet. Dia menulis atau menjadi penulis bersama lebih dari 200 RFC, termasuk protokol-protokol dasar seperti TCP, IP, dan SMTP.
- Selain kontribusi teknisnya, Postel dikenal karena pendekatan yang rendah hati dan kolaboratif. Dia percaya akan pentingnya mencapai konsensus dan bekerja sama untuk membangun jaringan yang kokoh dan interoperabel.
- Postel menjabat sebagai Direktur Divisi Jaringan Komputer di Information Sciences Institute (ISI) University of Southern California (USC) dari 1977 hingga kematiannya yang tiba-tiba pada 1998.
- Sebagai pengakuan atas kontribusinya yang luar biasa, Postel dianugerahi

penghargaan bergengsi Turing Award pada tahun 1998 secara anumerta, yang sering disebut sebagai “Nobel Prize dalam bidang Komputasi.”

Kelas JSONFixer mendorong ketangguhan, fleksibilitas, dan interoperabilitas, yang merupakan nilai-nilai inti yang dijunjung tinggi Postel sepanjang karirnya. Dengan membangun sistem yang toleran terhadap ketidaksempurnaan sambil tetap menjaga kepatuhan ketat terhadap protokol, kita dapat menciptakan aplikasi yang lebih tangguh dan adaptif dalam menghadapi tantangan dunia nyata.

Pertimbangan dan Kontraindikasi

Penerapan pendekatan data yang dapat memperbaiki diri sepenuhnya bergantung pada jenis data yang ditangani oleh aplikasi Anda. Ada alasan mengapa Anda mungkin tidak ingin sekadar melakukan monkeypatch pada `JSON.parse` untuk secara otomatis memperbaiki *semua kesalahan parsing JSON* dalam aplikasi Anda: tidak semua kesalahan dapat atau harus diperbaiki secara otomatis.

Perbaikan mandiri menjadi sangat rumit ketika dikaitkan dengan persyaratan regulasi atau kepatuhan terkait penanganan dan pemrosesan data. Beberapa industri, seperti kesehatan dan keuangan, memiliki regulasi yang sangat ketat mengenai integritas data dan kemampuan audit sehingga melakukan perbaikan data “kotak hitam” tanpa pengawasan atau pencatatan yang tepat dapat melanggar regulasi tersebut. Sangat penting untuk memastikan bahwa teknik data yang dapat memperbaiki diri yang Anda kembangkan selaras dengan kerangka hukum dan regulasi yang berlaku.

Penerapan teknik data yang dapat memperbaiki diri, terutama yang melibatkan model AI, juga dapat berdampak besar pada kinerja aplikasi dan pemanfaatan sumber daya. Memproses volume data yang besar melalui model AI untuk deteksi dan perbaikan kesalahan dapat memakan banyak sumber daya komputasi. Penting untuk menilai

pertimbangan untung-rugi antara manfaat data yang dapat memperbaiki diri dan biaya kinerja serta sumber daya yang terkait.

Dengan demikian, mari kita dalami faktor-faktor yang terlibat dalam menentukan kapan dan di mana menerapkan pendekatan yang ampuh ini.

Kritikalitas Data

Ketika mempertimbangkan penerapan teknik data yang dapat memperbaiki diri, sangat penting untuk menilai kritikalitas data yang sedang diproses. Tingkat kritikalitas mengacu pada pentingnya dan sensitivitas data dalam konteks aplikasi Anda dan domain bisnisnya.

Dalam beberapa kasus, memperbaiki kesalahan data secara otomatis mungkin tidak tepat, terutama jika data tersebut sangat sensitif atau memiliki implikasi hukum. Sebagai contoh, pertimbangkan skenario berikut:

1. **Transaksi Keuangan:** Dalam aplikasi keuangan, seperti sistem perbankan atau platform perdagangan, akurasi data sangat penting. Bahkan kesalahan kecil dalam data keuangan dapat memiliki konsekuensi signifikan, seperti saldo rekening yang tidak tepat, dana yang salah rute, atau keputusan perdagangan yang keliru. Dalam kasus ini, perbaikan otomatis tanpa verifikasi dan audit menyeluruh dapat menimbulkan risiko yang tidak dapat diterima.
2. **Catatan Medis:** Aplikasi kesehatan menangani data pasien yang sangat sensitif dan rahasia. Ketidakkuratan dalam catatan medis dapat memiliki implikasi serius bagi keselamatan pasien dan keputusan pengobatan. Memodifikasi data medis secara otomatis tanpa pengawasan dan validasi yang tepat oleh profesional kesehatan yang berkualifikasi dapat melanggar persyaratan regulasi dan membahayakan kesejahteraan pasien.
3. **Dokumen Hukum:** Aplikasi yang menangani dokumen hukum, seperti kontrak, perjanjian, atau berkas pengadilan, memerlukan akurasi dan integritas yang ketat.

Bahkan kesalahan kecil dalam data hukum dapat memiliki dampak hukum yang signifikan. Perbaikan otomatis dalam domain ini mungkin tidak tepat, karena data seringkali memerlukan tinjauan manual dan verifikasi oleh ahli hukum untuk memastikan keabsahan dan keberlakuannya.

Dalam skenario data kritis ini, risiko yang terkait dengan koreksi otomatis seringkali lebih besar daripada potensi manfaatnya. Konsekuensi dari memperkenalkan kesalahan atau memodifikasi data secara tidak tepat bisa sangat serius, yang mengakibatkan kerugian finansial, kewajiban hukum, atau bahkan membahayakan individu.

Ketika menangani data yang sangat kritis, sangat penting untuk memprioritaskan proses verifikasi dan validasi manual. Pengawasan dan keahlian manusia sangat penting dalam memastikan keakuratan dan integritas data. Teknik penyembuhan otomatis masih dapat digunakan untuk menandai potensi kesalahan atau inkonsistensi, tetapi keputusan akhir mengenai koreksi harus melibatkan pertimbangan dan persetujuan manusia.

Namun, perlu dicatat bahwa tidak semua data dalam suatu aplikasi memiliki tingkat kekritisannya yang sama. Dalam aplikasi yang sama, mungkin ada subset data yang kurang sensitif atau memiliki dampak yang lebih rendah jika terjadi kesalahan. Dalam kasus seperti ini, teknik penyembuhan data otomatis dapat diterapkan secara selektif pada subset data tertentu tersebut, sementara data kritis tetap harus melalui verifikasi manual.

Kuncinya adalah dengan cermat menilai tingkat kekritisannya setiap kategori data dalam aplikasi Anda dan menentukan pedoman serta proses yang jelas untuk menangani koreksi berdasarkan risiko dan implikasi yang terkait. Dengan membedakan antara data kritis (misalnya buku besar, catatan medis) dan data non-kritis (misalnya alamat surat, peringatan sumber daya), Anda dapat mencapai keseimbangan antara memanfaatkan kelebihan teknik penyembuhan data otomatis di tempat yang sesuai dan mempertahankan kontrol serta pengawasan yang ketat di tempat yang diperlukan.

Pada akhirnya, keputusan untuk menerapkan teknik penyembuhan data otomatis pada data kritis harus dibuat dengan berkonsultasi dengan pakar domain, penasihat hukum, dan pemangku kepentingan terkait lainnya. Sangat penting untuk mempertimbangkan

persyaratan khusus, regulasi, dan risiko yang terkait dengan data aplikasi Anda dan menyelaraskan strategi koreksi data yang sesuai.

Tingkat Keparahan Kesalahan

Ketika menerapkan teknik penyembuhan data otomatis, penting untuk menilai tingkat keparahan dan dampak dari kesalahan data. Tidak semua kesalahan diciptakan setara, dan tindakan yang tepat mungkin berbeda tergantung pada tingkat keparahan masalahnya.

Inkonsistensi kecil atau masalah pemformatan mungkin cocok untuk koreksi otomatis. Misalnya, pekerja penyembuhan data yang ditugaskan untuk memperbaiki JSON yang rusak dapat menangani koma yang hilang atau tanda kutip ganda yang tidak di-escape tanpa mengubah makna atau struktur data secara signifikan. Jenis kesalahan seperti ini seringkali mudah diperbaiki dan memiliki dampak minimal terhadap integritas data secara keseluruhan.

Namun, kesalahan yang lebih serius yang secara fundamental mengubah makna atau integritas data mungkin memerlukan pendekatan yang berbeda. Dalam kasus seperti ini, koreksi otomatis mungkin tidak mencukupi, dan intervensi manusia mungkin diperlukan untuk memastikan keakuratan dan validitas data.

Di sinilah konsep penggunaan AI itu sendiri berperan untuk membantu menentukan tingkat keparahan kesalahan. Dengan memanfaatkan kemampuan model AI, kita dapat merancang pekerja data yang dapat memperbaiki diri yang tidak hanya memperbaiki kesalahan tetapi juga menilai tingkat keparahan kesalahan tersebut dan membuat keputusan yang tepat tentang cara menanganinya.

Sebagai contoh, mari kita pertimbangkan sebuah pekerja data yang dapat memperbaiki diri yang bertanggung jawab untuk mengoreksi ketidakkonsistenan dalam aliran data ke basis data pelanggan. Pekerja tersebut dapat dirancang untuk menganalisis data dan mengidentifikasi potensi kesalahan, seperti informasi yang hilang atau bertentangan.

Namun, alih-alih secara otomatis memperbaiki semua kesalahan, pekerja tersebut dapat dilengkapi dengan pemanggilan alat tambahan yang memungkinkannya menandai kesalahan serius untuk peninjauan manusia.

Berikut contoh bagaimana hal ini dapat diimplementasikan:

```
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```\n" }
27
28    self.stop = ["```\n"]
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**

Dalam contoh ini, worker `CustomerDataHealer` dirancang untuk mengidentifikasi dan memperbaiki inkonsistensi dalam data pelanggan. Sekali lagi, kita menggunakan `Pembatasan Respons` dan `Ventriloquist` untuk mendapatkan output terstruktur. Yang penting, direktif sistem worker mencakup instruksi untuk menggunakan fungsi `flag_for_review` jika ditemukan kesalahan yang serius.

Ketika worker memproses data pelanggan, ia menganalisis data dan berusaha memperbaiki setiap inkonsistensi. Jika worker menentukan bahwa kesalahan tersebut serius dan memerlukan intervensi manusia, ia dapat menggunakan tool `flag_for_review` untuk menandai data tersebut dan memberikan alasan penandaan.

Metode `chat_completion` dipanggil dengan `json: true` untuk mengurai data pelanggan yang telah dikoreksi sebagai JSON. Tidak ada ketentuan untuk melakukan pengulangan setelah pemanggilan fungsi, sehingga hasilnya akan kosong jika `flag_for_review` dijalankan. Jika tidak, data pelanggan diperbarui dengan data yang telah ditinjau dan berpotensi dikoreksi.

Dengan memasukkan penilaian tingkat keparahan kesalahan dan opsi untuk menandai data untuk ditinjau manusia, worker data yang dapat memperbaiki diri menjadi lebih cerdas dan adaptif. Ia dapat menangani kesalahan minor secara otomatis sambil meningkatkan kesalahan serius kepada ahli manusia untuk intervensi manual.

Kriteria spesifik untuk menentukan tingkat keparahan kesalahan dapat didefinisikan dalam direktif worker berdasarkan pengetahuan domain dan kebutuhan bisnis. Faktor-faktor seperti dampak terhadap integritas data, potensi kehilangan atau kerusakan data, dan konsekuensi dari data yang tidak tepat dapat dipertimbangkan saat menilai tingkat keparahan.

Dengan memanfaatkan AI untuk menilai tingkat keparahan kesalahan dan menyediakan opsi untuk intervensi manusia, teknik data yang dapat memperbaiki diri dapat mencapai keseimbangan antara otomatisasi dan mempertahankan akurasi data. Pen-

dekatan ini memastikan bahwa kesalahan minor diperbaiki secara efisien sementara kesalahan serius mendapat perhatian dan keahlian yang diperlukan dari peninjau manusia.

Kompleksitas Domain

Ketika mempertimbangkan penerapan teknik data yang dapat memperbaiki diri, penting untuk mengevaluasi kompleksitas domain data dan aturan yang mengatur struktur serta hubungannya. Kompleksitas domain dapat secara signifikan mempengaruhi efektivitas dan kelayakan pendekatan koreksi data otomatis.

Teknik data yang dapat memperbaiki diri bekerja dengan baik ketika data mengikuti pola dan batasan yang terdefinisi dengan baik. Dalam domain di mana struktur data relatif sederhana dan hubungan antar elemen data bersifat lugas, koreksi otomatis dapat diterapkan dengan tingkat kepercayaan yang tinggi. Sebagai contoh, memperbaiki masalah pemformatan atau menerapkan batasan tipe data dasar seringkali dapat ditangani secara efektif oleh worker data yang dapat memperbaiki diri.

Namun, seiring meningkatnya kompleksitas domain data, tantangan yang terkait dengan koreksi data otomatis juga bertambah. Dalam domain dengan logika bisnis yang rumit, hubungan kompleks antara entitas data, atau aturan dan pengecualian khusus domain, teknik data yang dapat memperbaiki diri mungkin tidak selalu dapat menangkap nuansa tersebut dan dapat menimbulkan konsekuensi yang tidak diinginkan.

Mari kita lihat contoh domain yang kompleks: sistem perdagangan keuangan. Dalam domain ini, data melibatkan berbagai instrumen keuangan, data pasar, aturan perdagangan, dan persyaratan regulasi. Hubungan antara elemen data yang berbeda bisa sangat rumit, dan aturan yang mengatur validitas dan konsistensi data dapat sangat spesifik untuk domain tersebut.

Dalam domain yang kompleks seperti ini, pekerja data yang dapat memperbaiki diri yang bertugas mengoreksi inkonsistensi dalam data perdagangan perlu memiliki pemahaman mendalam tentang aturan dan batasan khusus domain. Pekerja tersebut perlu

mempertimbangkan faktor-faktor seperti regulasi pasar, batas perdagangan, perhitungan risiko, dan prosedur penyelesaian. Koreksi otomatis dalam konteks ini mungkin tidak selalu dapat menangkap kompleksitas domain secara penuh dan mungkin secara tidak sengaja memunculkan kesalahan atau melanggar aturan khusus domain.

Untuk mengatasi tantangan kompleksitas domain, teknik data yang dapat memperbaiki diri dapat ditingkatkan dengan memasukkan pengetahuan dan aturan khusus domain ke dalam model AI dan para pekerja. Hal ini dapat dicapai melalui teknik-teknik seperti:

1. **Pelatihan Khusus Domain:** Model AI yang digunakan untuk data yang dapat memperbaiki diri dapat diarahkan atau bahkan disesuaikan pada dataset khusus domain yang menangkap kerumitan dan aturan domain tertentu. Dengan mengekspos model pada data dan skenario yang representatif, mereka dapat mempelajari pola, batasan, dan pengecualian yang spesifik untuk domain tersebut.
2. **Batasan Berbasis Aturan:** Pekerja data yang dapat memperbaiki diri dapat ditingkatkan dengan batasan berbasis aturan eksplisit yang mengkodekan pengetahuan khusus domain. Aturan-aturan ini dapat didefinisikan oleh para ahli domain dan diintegrasikan ke dalam proses koreksi data. Model AI kemudian dapat menggunakan aturan-aturan ini untuk memandu keputusan mereka dan memastikan kepatuhan terhadap persyaratan khusus domain.
3. **Kolaborasi dengan Ahli Domain:** Dalam domain yang kompleks, sangat penting untuk melibatkan ahli domain dalam desain dan pengembangan teknik data yang dapat memperbaiki diri. Ahli domain dapat memberikan wawasan berharga tentang kerumitan data, aturan bisnis, dan kasus-kasus ekstrem yang mungkin terjadi. Pengetahuan mereka dapat dimasukkan ke dalam model AI dan para pekerja untuk meningkatkan akurasi dan keandalan koreksi data otomatis menggunakan pola [Human In The Loop](#).
4. **Pendekatan Bertahap dan Iteratif:** Ketika berhadapan dengan domain yang kompleks, seringkali bermanfaat untuk mengadopsi pendekatan bertahap dan iteratif untuk data yang dapat memperbaiki diri. Alih-alih mencoba mengotomatisasi koreksi untuk seluruh domain sekaligus, fokus pada subdomain atau

kategori data tertentu di mana aturan dan batasannya dipahami dengan baik. Secara bertahap perluas cakupan teknik yang dapat memperbaiki diri seiring bertambahnya pemahaman tentang domain dan terbuktinya efektivitas teknik tersebut.

Dengan mempertimbangkan kompleksitas domain data dan memasukkan pengetahuan spesifik domain ke dalam teknik data yang dapat memperbaiki diri, Anda dapat mencapai keseimbangan antara otomatisasi dan akurasi. Penting untuk menyadari bahwa data yang dapat memperbaiki diri bukanlah solusi yang cocok untuk semua situasi dan pendekatan harus disesuaikan dengan persyaratan dan tantangan spesifik setiap domain.

Dalam domain yang kompleks, pendekatan hibrida yang menggabungkan teknik data yang dapat memperbaiki diri dengan keahlian dan pengawasan manusia dapat menjadi yang paling efektif. Koreksi otomatis dapat menangani kasus-kasus rutin dan yang telah terdefinisi dengan baik, sementara skenario kompleks atau pengecualian dapat ditandai untuk ditinjau dan diintervensi oleh manusia. Pendekatan kolaboratif ini memastikan bahwa manfaat otomatisasi dapat direalisasikan sambil mempertahankan kontrol dan akurasi yang diperlukan dalam domain data yang kompleks.

Kemampuan Penjelasan dan Transparansi

Kemampuan penjelasan mengacu pada kemampuan untuk memahami dan menginterpretasikan penalaran di balik keputusan yang dibuat oleh model AI, sementara transparansi melibatkan penyediaan visibilitas yang jelas dalam proses koreksi data.

Dalam banyak konteks, modifikasi data perlu dapat diaudit dan dipertanggungjawabkan. Para pemangku kepentingan, termasuk pengguna bisnis, auditor, dan badan regulasi, mungkin memerlukan penjelasan mengapa koreksi data tertentu dilakukan dan bagaimana model AI sampai pada keputusan tersebut. Hal ini sangat penting dalam domain di mana akurasi dan integritas data memiliki implikasi signifikan, seperti keuangan, kesehatan, dan masalah hukum.

Untuk mengatasi kebutuhan akan kemampuan penjelasan dan transparansi, teknik data yang dapat memperbaiki diri harus memasukkan mekanisme yang memberikan wawasan tentang proses pengambilan keputusan model AI. Ini dapat dicapai melalui berbagai pendekatan:

1. **Rantai Pemikiran:** Meminta model untuk menjelaskan pemikirannya “secara terbuka” sebelum menerapkan perubahan pada data dapat memungkinkan pemahaman yang lebih mudah tentang proses pengambilan keputusan dan dapat menghasilkan penjelasan yang dapat dibaca manusia untuk koreksi yang dilakukan. Timbal baliknya adalah sedikit lebih kompleks dalam memisahkan penjelasan dari output data terstruktur, yang dapat diatasi dengan...
2. **Pembuatan Penjelasan:** Pekerja data yang dapat memperbaiki diri dapat dilengkapi dengan kemampuan untuk menghasilkan penjelasan yang dapat dibaca manusia untuk koreksi yang mereka buat. Ini dapat dicapai dengan meminta model untuk menghasilkan proses pengambilan keputusannya sebagai penjelasan yang mudah dimengerti *yang terintegrasi ke dalam data itu sendiri*. Misalnya, pekerja data yang dapat memperbaiki diri dapat menghasilkan laporan yang menyoroti ketidakkonsistenan data spesifik yang diidentifikasi, koreksi yang diterapkan, dan alasan di balik koreksi tersebut.
3. **Tingkat Kepentingan Fitur:** Model AI dapat diinstruksikan dengan informasi tentang pentingnya berbagai fitur atau atribut dalam proses koreksi data sebagai bagian dari arahan mereka. Arahan tersebut, pada gilirannya, dapat diperlihatkan kepada pemangku kepentingan manusia. Dengan mengidentifikasi faktor-faktor kunci yang mempengaruhi keputusan model, pemangku kepentingan dapat memperoleh wawasan tentang penalaran di balik koreksi dan menilai validitasnya.
4. **Pencatatan dan Audit:** Penerapan mekanisme pencatatan dan audit yang komprehensif sangat penting untuk menjaga transparansi dalam proses data penyempurnaan mandiri. Setiap koreksi data yang dilakukan oleh model AI harus dicatat, termasuk data asli, data yang dikoreksi, dan tindakan spesifik yang diambil. Jejak

audit ini memungkinkan analisis retrospektif dan memberikan catatan yang jelas tentang modifikasi yang dilakukan pada data.

5. **Pendekatan Manusia dalam Siklus:** Memasukkan pendekatan manusia dalam siklus dapat meningkatkan kemampuan penjelasan dan transparansi teknik data penyembuhan mandiri. Dengan melibatkan para ahli dalam peninjauan dan validasi koreksi yang dihasilkan AI, organisasi dapat memastikan bahwa koreksi tersebut selaras dengan pengetahuan domain dan kebutuhan bisnis. Pengawasan manusia menambahkan lapisan akuntabilitas tambahan dan memungkinkan identifikasi potensi bias atau kesalahan dalam model AI.
6. **Pemantauan dan Evaluasi Berkelanjutan:** Pemantauan dan evaluasi rutin terhadap kinerja teknik data penyembuhan mandiri sangat penting untuk menjaga transparansi dan kepercayaan. Dengan menilai akurasi dan efektivitas model AI dari waktu ke waktu, organisasi dapat mengidentifikasi penyimpangan atau anomali dan mengambil tindakan korektif. Pemantauan berkelanjutan membantu memastikan bahwa proses data penyembuhan mandiri tetap dapat diandalkan dan selaras dengan hasil yang diinginkan.

Kemampuan penjelasan dan transparansi adalah pertimbangan penting ketika menerapkan teknik data penyembuhan mandiri. Dengan memberikan penjelasan yang jelas untuk koreksi data, mempertahankan jejak audit yang komprehensif, dan melibatkan pengawasan manusia, organisasi dapat membangun kepercayaan dalam proses data penyembuhan mandiri dan memastikan bahwa modifikasi yang dilakukan pada data dapat dipertanggungjawabkan dan selaras dengan tujuan bisnis.

Penting untuk menjaga keseimbangan antara manfaat otomatisasi dan kebutuhan akan transparansi. Meskipun teknik data penyembuhan mandiri dapat secara signifikan meningkatkan kualitas dan efisiensi data, hal ini tidak boleh mengorbankan visibilitas dan kontrol atas proses koreksi data. Dengan merancang pekerja data penyembuhan mandiri dengan mempertimbangkan kemampuan penjelasan dan transparansi, organisasi dapat memanfaatkan kekuatan AI sambil mempertahankan tingkat akuntabilitas dan kepercayaan yang diperlukan dalam data.

Konsekuensi yang Tidak Diinginkan

Meskipun teknik data penyembuhan mandiri bertujuan untuk meningkatkan kualitas dan konsistensi data, penting untuk menyadari potensi konsekuensi yang tidak diinginkan. Koreksi otomatis, jika tidak dirancang dan dipantau dengan hati-hati, dapat secara tidak sengaja mengubah makna atau konteks data, yang mengarah pada masalah hilir.

Salah satu risiko utama dari data penyembuhan mandiri adalah masuknya bias atau kesalahan dalam proses koreksi data. Model AI, seperti sistem perangkat lunak lainnya, dapat dipengaruhi oleh bias yang ada dalam data pelatihan atau yang diperkenalkan melalui desain algoritma. Jika bias ini tidak diidentifikasi dan dimitigasi, mereka dapat menyebar melalui proses data penyembuhan mandiri dan menghasilkan modifikasi data yang miring atau tidak tepat.

Sebagai contoh, pertimbangkan sebuah pekerja data swapulih yang bertugas memperbaiki inkonsistensi dalam data demografis pelanggan. Jika model AI telah mempelajari bias dari data historis, seperti mengaitkan pekerjaan atau tingkat pendapatan tertentu dengan gender atau etnis tertentu, model tersebut mungkin membuat asumsi yang salah dan memodifikasi data dengan cara yang justru memperkuat bias tersebut. Hal ini dapat mengakibatkan profil pelanggan yang tidak akurat, keputusan bisnis yang keliru, dan berpotensi menghasilkan dampak yang diskriminatif.

Konsekuensi yang tidak diinginkan lainnya adalah hilangnya informasi atau konteks berharga selama proses perbaikan data. Teknik data swapulih seringkali berfokus pada standarisasi dan normalisasi data untuk memastikan konsistensi. Namun, dalam beberapa kasus, data asli mungkin mengandung nuansa, pengecualian, atau informasi kontekstual yang penting untuk memahami gambaran secara utuh. Perbaikan otomatis yang memberlakukan standarisasi secara membabi buta dapat secara tidak sengaja menghilangkan atau mengaburkan informasi berharga ini.

Misalnya, bayangkan seorang pekerja data swapulih yang bertanggung jawab untuk

memperbaiki inkonsistensi dalam catatan medis. Jika pekerja tersebut menemukan riwayat medis pasien dengan kondisi langka atau rencana pengobatan yang tidak biasa, ia mungkin mencoba menormalisasi data tersebut agar sesuai dengan pola yang lebih umum. Namun, dalam prosesnya, ia mungkin kehilangan detail dan konteks spesifik yang sangat penting untuk merepresentasikan situasi unik pasien secara akurat. Kehilangan informasi ini dapat memiliki implikasi serius bagi perawatan pasien dan pengambilan keputusan medis.

Untuk mengurangi risiko konsekuensi yang tidak diinginkan, penting untuk mengambil pendekatan proaktif saat merancang dan menerapkan teknik data swapulih:

1. **Pengujian dan Validasi Menyeluruh:** Sebelum menerapkan pekerja data swapulih dalam produksi, sangat penting untuk menguji dan memvalidasi perilaku mereka secara menyeluruh terhadap berbagai skenario. Ini termasuk pengujian dengan dataset representatif yang mencakup berbagai kasus ekstrem, pengecualian, dan potensi bias. Pengujian yang ketat membantu mengidentifikasi dan mengatasi konsekuensi yang tidak diinginkan sebelum berdampak pada data dunia nyata.
2. **Pemantauan dan Evaluasi Berkelanjutan:** Menerapkan mekanisme pemantauan dan evaluasi berkelanjutan sangat penting untuk mendeteksi dan mengurangi konsekuensi yang tidak diinginkan dari waktu ke waktu. Meninjau hasil proses data swapulih secara rutin, menganalisis dampak pada sistem hilir dan pengambilan keputusan, serta mengumpulkan umpan balik dari pemangku kepentingan dapat membantu mengidentifikasi efek merugikan dan memicu tindakan korektif yang tepat waktu. Jika organisasi Anda memiliki dasbor operasional, menambahkan metrik yang jelas terkait perubahan data otomatis mungkin adalah ide yang bagus. Menambahkan alarm yang terhubung dengan penyimpangan besar dari aktivitas perubahan data normal mungkin adalah ide yang lebih baik lagi!
3. **Pengawasan dan Intervensi Manusia:** Mempertahankan pengawasan manusia dan kemampuan untuk melakukan intervensi dalam proses data swapulih sangat

penting. Meskipun otomatisasi dapat sangat meningkatkan efisiensi, penting untuk memiliki ahli manusia yang meninjau dan memvalidasi koreksi yang dibuat oleh model AI, terutama dalam domain yang kritis atau sensitif. Penilaian manusia dan keahlian bidang dapat membantu mengidentifikasi dan mengatasi konsekuensi yang tidak diinginkan yang mungkin muncul.

4. **Explainable AI (XAI) dan Transparansi:** Seperti yang dibahas pada subbagian sebelumnya, penerapan teknik Explainable AI dan memastikan transparansi dalam proses data yang dapat memperbaiki diri dapat membantu mengurangi konsekuensi yang tidak diinginkan. Dengan memberikan penjelasan yang jelas untuk koreksi data dan memelihara jejak audit yang komprehensif, organisasi dapat lebih memahami dan melacak alasan di balik modifikasi yang dilakukan oleh model AI.
5. **Pendekatan Bertahap dan Iteratif:** Mengadopsi pendekatan bertahap dan iteratif untuk data yang dapat memperbaiki diri dapat membantu meminimalkan risiko konsekuensi yang tidak diinginkan. Alih-alih menerapkan koreksi otomatis pada seluruh kumpulan data sekaligus, mulailah dengan subset data dan perluas cakupannya secara bertahap seiring teknik terbukti efektif dan dapat diandalkan. Hal ini memungkinkan pemantauan dan penyesuaian yang cermat sepanjang prosesnya, mengurangi dampak dari konsekuensi yang tidak diinginkan.
6. **Kolaborasi dan Umpan Balik:** Melibatkan pemangku kepentingan dari berbagai domain dan mendorong kolaborasi serta umpan balik selama proses data yang dapat memperbaiki diri dapat membantu mengidentifikasi dan mengatasi konsekuensi yang tidak diinginkan. Secara rutin mencari masukan dari ahli domain, pengguna data, dan pengguna akhir dapat memberikan wawasan berharga tentang dampak nyata dari koreksi data dan menyoroti masalah yang mungkin terlewatkan.

Dengan secara proaktif mengatasi risiko konsekuensi yang tidak diinginkan dan menerapkan pengamanan yang tepat, organisasi dapat memanfaatkan keuntungan dari

teknik data yang dapat memperbaiki diri sambil meminimalkan potensi efek merugikan. Penting untuk mendekati data yang dapat memperbaiki diri sebagai proses iteratif dan kolaboratif, terus memantau, mengevaluasi, dan menyempurnakan teknik untuk memastikan selaras dengan hasil yang diinginkan dan mempertahankan integritas serta keandalan data.

Ketika mempertimbangkan penggunaan pola data yang dapat memperbaiki diri, penting untuk mengevaluasi faktor-faktor ini dengan cermat dan menimbang manfaat terhadap potensi risiko dan keterbatasan. Dalam beberapa kasus, pendekatan hibrid yang menggabungkan koreksi otomatis dengan pengawasan dan intervensi manusia mungkin menjadi solusi yang paling tepat.

Perlu juga dicatat bahwa teknik data yang dapat memperbaiki diri tidak boleh dianggap sebagai pengganti untuk validasi data yang kuat, sanitasi input, dan mekanisme penanganan kesalahan. Praktik-praktik mendasar ini tetap penting untuk memastikan integritas dan keamanan data. Data yang dapat memperbaiki diri harus dipandang sebagai pendekatan pelengkap yang dapat menambah dan meningkatkan langkah-langkah yang sudah ada.

Pada akhirnya, keputusan untuk menggunakan pola data yang dapat memperbaiki diri bergantung pada persyaratan, batasan, dan prioritas spesifik aplikasi Anda. Dengan mempertimbangkan dengan cermat pertimbangan yang diuraikan di atas dan menyelaraskannya dengan tujuan dan arsitektur aplikasi Anda, Anda dapat membuat keputusan yang tepat tentang kapan dan bagaimana memanfaatkan teknik data yang dapat memperbaiki diri secara efektif.

Pembuatan Konten Kontekstual



Pola Pembuatan Konten Kontekstual memanfaatkan kekuatan model bahasa besar (LLM) untuk menghasilkan konten dinamis dan kontekstual dalam aplikasi. Kategori pola ini mengakui pentingnya memberikan konten yang dipersonalisasi dan relevan kepada pengguna berdasarkan kebutuhan spesifik, preferensi, dan bahkan interaksi sebelumnya serta saat ini dengan aplikasi.

Dalam konteks pendekatan ini, “konten” mengacu pada konten primer (seperti posting blog, artikel, dll) dan meta-konten, seperti rekomendasi untuk konten primer.

Pola Pembuatan Konten Kontekstual dapat berperan penting dalam meningkatkan tingkat keterlibatan pengguna Anda, memberikan pengalaman yang disesuaikan, dan mengotomatisasi tugas pembuatan konten baik untuk Anda maupun pengguna Anda.

Dengan menggunakan pola yang kami jelaskan dalam bab ini, Anda dapat membuat aplikasi yang menghasilkan konten secara dinamis, beradaptasi dengan konteks dan masukan secara real-time.

Pola-pola ini bekerja dengan mengintegrasikan LLM ke dalam keluaran aplikasi, mulai dari antarmuka pengguna (terkadang disebut sebagai “chrome”), email dan bentuk notifikasi lainnya, serta berbagai alur pembuatan konten.

Ketika pengguna berinteraksi dengan aplikasi atau memicu permintaan konten tertentu, aplikasi menangkap konteks yang relevan, seperti preferensi pengguna, interaksi sebelumnya, atau prompt tertentu. Informasi kontekstual ini kemudian dimasukkan ke dalam LLM, bersama dengan template atau panduan yang diperlukan dan digunakan untuk menghasilkan keluaran tekstual yang sebaliknya harus dikodekan secara manual, disimpan dalam database, atau dihasilkan secara algoritmik.

Konten yang dihasilkan LLM dapat mengambil berbagai bentuk, seperti rekomendasi yang dipersonalisasi, deskripsi produk dinamis, respons email yang disesuaikan, atau bahkan artikel atau posting blog lengkap. Salah satu penggunaan paling radikal dari konten ini yang saya rintis lebih dari setahun yang lalu adalah menghasilkan elemen UI secara dinamis seperti label formulir, tooltips, dan berbagai jenis teks penjelasan lainnya.

Personalisasi

Salah satu manfaat utama dari pola Pembuatan Konten Kontekstual adalah kemampuan untuk memberikan pengalaman yang sangat dipersonalisasi kepada pengguna. Dengan menghasilkan konten berdasarkan konteks spesifik pengguna, pola-pola ini memungkinkan aplikasi untuk menyesuaikan konten dengan minat, preferensi, dan interaksi masing-masing pengguna.

Personalisasi lebih dari sekadar menyisipkan nama pengguna ke dalam konten umum. Ini melibatkan pemanfaatan konteks yang kaya tentang setiap pengguna untuk meng-

hasilkan konten yang beresonansi dengan kebutuhan dan keinginan spesifik mereka. Konteks ini dapat mencakup berbagai faktor, seperti:

1. **Informasi Profil Pengguna:** Pada tingkat paling umum dalam penerapan teknik ini, data demografis, minat, preferensi, dan atribut profil lainnya dapat digunakan untuk menghasilkan konten yang selaras dengan latar belakang dan karakteristik pengguna.
2. **Data Perilaku:** Interaksi pengguna sebelumnya dengan aplikasi, seperti halaman yang dilihat, tautan yang diklik, atau produk yang dibeli, dapat memberikan wawasan berharga tentang perilaku dan minat mereka. Data ini dapat digunakan untuk menghasilkan saran konten yang mencerminkan pola keterlibatan mereka dan memprediksi kebutuhan masa depan mereka.
3. **Faktor Kontekstual:** Konteks pengguna saat ini, seperti lokasi mereka, perangkat, waktu hari, atau bahkan cuaca, dapat memengaruhi proses pembuatan konten. Misalnya, aplikasi perjalanan mungkin memiliki pekerja AI yang mampu menghasilkan rekomendasi yang dipersonalisasi berdasarkan lokasi pengguna saat ini dan kondisi cuaca yang berlaku.

Dengan memanfaatkan faktor-faktor kontekstual ini, pola Pembuatan Konten Kontekstual memungkinkan aplikasi untuk menyampaikan konten yang terasa dibuat khusus untuk setiap pengguna individual. Tingkat personalisasi ini memiliki beberapa manfaat signifikan:

1. **Peningkatan Keterlibatan:** Konten yang dipersonalisasi menarik perhatian pengguna dan membuat mereka tetap terlibat dengan aplikasi. Ketika pengguna merasa bahwa konten tersebut relevan dan berbicara langsung kepada kebutuhan mereka, mereka lebih cenderung menghabiskan lebih banyak waktu berinteraksi dengan aplikasi dan menjelajahi fitur-fiturnya.
2. **Peningkatan Kepuasan Pengguna:** Konten yang dipersonalisasi menunjukkan bahwa aplikasi memahami dan peduli tentang kebutuhan unik pengguna. Den-

gan menyediakan konten yang bermanfaat, informatif, dan selaras dengan minat mereka, aplikasi dapat meningkatkan kepuasan pengguna dan membangun hubungan yang lebih kuat dengan penggunanya.

3. **Tingkat Konversi yang Lebih Tinggi:** Dalam konteks aplikasi e-commerce atau pemasaran, konten yang dipersonalisasi dapat berdampak signifikan pada tingkat konversi. Dengan menampilkan produk, penawaran, atau rekomendasi yang disesuaikan dengan preferensi dan perilaku pengguna, aplikasi dapat meningkatkan kemungkinan pengguna mengambil tindakan yang diinginkan, seperti melakukan pembelian atau mendaftar layanan.

Produktivitas

Pola Pembuatan Konten Kontekstual dapat secara signifikan meningkatkan jenis-jenis produktivitas tertentu dengan mengurangi kebutuhan akan pembuatan dan pengeditan konten manual dalam proses kreatif. Dengan memanfaatkan kekuatan LLM, Anda dapat menghasilkan konten berkualitas tinggi dalam skala besar, menghemat waktu dan usaha yang seharusnya dihabiskan oleh pembuat konten dan pengembang Anda untuk melakukan pekerjaan manual yang membosankan.

Secara tradisional, para pembuat konten perlu melakukan riset, menulis, mengedit, dan memformat konten untuk memastikan bahwa konten tersebut memenuhi persyaratan aplikasi dan harapan pengguna. Proses ini bisa memakan waktu dan membutuhkan banyak sumber daya, terutama ketika volume konten semakin bertambah.

Namun, dengan pola Pembuatan Konten Kontekstual, proses pembuatan konten dapat diotomatisasi secara luas. LLM dapat menghasilkan konten yang koheren, benar secara tata bahasa, dan relevan secara kontekstual berdasarkan prompt dan pedoman yang diberikan. Otomatisasi ini menawarkan beberapa manfaat produktivitas:

1. **Mengurangi Upaya Manual:** Dengan mendelegasikan tugas pembuatan konten kepada LLM, pembuat konten dapat fokus pada tugas-tugas tingkat tinggi seperti

strategi konten, pencarian ide, dan jaminan kualitas. Mereka dapat memberikan konteks, template, dan pedoman yang diperlukan kepada LLM dan membiarkannya menangani pembuatan konten yang sebenarnya. Hal ini mengurangi upaya manual yang diperlukan untuk menulis dan mengedit, memungkinkan pembuatan konten menjadi lebih produktif dan efisien.

2. **Pembuatan Konten Lebih Cepat:** LLM dapat menghasilkan konten jauh lebih cepat dibandingkan penulis manusia. Dengan prompt dan pedoman yang tepat, LLM dapat menghasilkan beberapa konten dalam hitungan detik atau menit. Kecepatan ini memungkinkan aplikasi untuk menghasilkan konten dengan kecepatan yang jauh lebih tinggi, mengimbangi tuntutan pengguna dan lanskap digital yang terus berubah.

Apakah pembuatan konten yang lebih cepat mengarah pada situasi “tragedi milik bersama” di mana internet dibanjiri konten yang tidak dibaca siapa pun? Sayangnya, saya menduga jawabannya adalah ya.

3. **Konsistensi dan Kualitas:** LLM dapat dengan mudah merevisi konten sehingga konsisten dalam gaya, nada, dan kualitas. Dengan pedoman dan contoh yang jelas, jenis aplikasi tertentu (misalnya ruang redaksi, PR, dll.) dapat memastikan bahwa konten yang dihasilkan manusia selaras dengan suara merek mereka dan memenuhi standar kualitas yang diinginkan. Konsistensi ini mengurangi kebutuhan untuk editing dan revisi yang ekstensif, menghemat waktu dan upaya dalam proses pembuatan konten.
4. **Iterasi dan Optimisasi:** Pola Pembuatan Konten Kontekstual memungkinkan iterasi dan optimisasi konten yang cepat. Dengan menyesuaikan prompt, template, atau pedoman yang diberikan kepada LLM, aplikasi Anda dapat dengan cepat menghasilkan variasi konten dan menguji pendekatan berbeda secara

otomatis yang tidak pernah mungkin dilakukan di masa lalu. Proses iteratif ini memungkinkan eksperimen yang lebih cepat dan penyempurnaan strategi konten, yang menghasilkan konten yang lebih efektif dan menarik seiring waktu. Teknik khusus ini bisa menjadi pengubah permainan total untuk aplikasi seperti e-commerce yang hidup dan mati berdasarkan tingkat pentalan dan keterlibatan



Penting untuk dicatat bahwa meskipun pola Pembuatan Konten Kontekstual dapat sangat meningkatkan produktivitas, hal ini tidak sepenuhnya menghilangkan kebutuhan akan keterlibatan manusia. Kreator konten dan editor tetap memainkan peran penting dalam menentukan strategi konten secara keseluruhan, memberikan panduan kepada LLM, dan memastikan kualitas serta kesesuaian konten yang dihasilkan.

Dengan mengotomatisasi aspek pembuatan konten yang lebih berulang dan memakan waktu, pola Pembuatan Konten Kontekstual membebaskan waktu dan sumber daya manusia yang berharga yang dapat dialihkan ke tugas-tugas bernilai lebih tinggi. Peningkatan produktivitas ini memungkinkan Anda untuk memberikan konten yang lebih personal dan menarik kepada pengguna sambil mengoptimalkan alur kerja pembuatan konten.

Iterasi dan Eksperimen Cepat

Pola Pembuatan Konten Kontekstual memungkinkan Anda untuk dengan cepat melakukan iterasi dan bereksperimen dengan variasi konten yang berbeda, memungkinkan optimasi dan penyempurnaan strategi konten Anda lebih cepat. Anda dapat menghasilkan beberapa versi konten dalam hitungan detik, cukup dengan menyesuaikan konteks, template, atau panduan yang diberikan kepada model.

Kemampuan iterasi cepat ini menawarkan beberapa manfaat utama:

1. **Pengujian dan Optimasi:** Dengan kemampuan untuk menghasilkan variasi konten dengan cepat, Anda dapat dengan mudah menguji pendekatan berbeda dan mengukur efektivitasnya. Misalnya, Anda dapat menghasilkan beberapa versi deskripsi produk atau pesan pemasaran, masing-masing disesuaikan dengan segmen pengguna atau konteks tertentu. Dengan menganalisis metrik keterlibatan pengguna, seperti tingkat klik atau tingkat konversi, Anda dapat mengidentifikasi variasi konten yang paling efektif dan mengoptimalkan strategi konten Anda dengan tepat.
2. **Pengujian A/B:** Pola Pembuatan Konten Kontekstual memungkinkan pengujian A/B konten yang mulus. Anda dapat menghasilkan dua atau lebih variasi konten dan menyajikannya secara acak ke kelompok pengguna yang berbeda. Dengan membandingkan kinerja setiap variasi, Anda dapat menentukan konten mana yang paling sesuai dengan target audiens Anda. Pendekatan berbasis data ini memungkinkan Anda untuk membuat keputusan yang tepat dan terus menyempurnakan konten Anda untuk memaksimalkan keterlibatan pengguna dan mencapai hasil yang diinginkan.
3. **Eksperimen Personalisasi:** Iterasi dan eksperimen cepat sangat berharga ketika berkaitan dengan personalisasi. Dengan pola Pembuatan Konten Kontekstual, Anda dapat dengan cepat menghasilkan variasi konten yang dipersonalisasi berdasarkan segmen pengguna, preferensi, atau perilaku yang berbeda. Dengan bereksperimen dengan strategi personalisasi yang berbeda, Anda dapat mengidentifikasi pendekatan yang paling efektif untuk melibatkan pengguna individual dan memberikan pengalaman yang disesuaikan.
4. **Beradaptasi dengan Tren yang Berubah:** Kemampuan untuk melakukan iterasi dan bereksperimen dengan cepat memungkinkan Anda untuk tetap tangkas dan beradaptasi dengan tren dan preferensi pengguna yang berubah. Ketika topik, kata kunci, atau perilaku pengguna baru muncul, Anda dapat dengan cepat menghasilkan konten yang selaras dengan tren tersebut. Dengan terus bereksperimen dan menyempurnakan konten Anda, Anda dapat tetap relevan

dan mempertahankan keunggulan kompetitif dalam lanskap digital yang terus berkembang.

5. **Eksperimen Hemat Biaya:** Eksperimen konten tradisional seringkali membutuhkan waktu dan sumber daya yang signifikan, karena pembuat konten perlu mengembangkan dan menguji berbagai variasi secara manual. Namun, dengan pola Pembuatan Konten Kontekstual, biaya eksperimen sangat berkurang. LLM dapat menghasilkan variasi konten dengan cepat dan dalam skala besar, memungkinkan Anda untuk mengeksplorasi berbagai ide dan pendekatan tanpa mengeluarkan biaya yang besar.

Untuk memaksimalkan iterasi dan eksperimen yang cepat, penting untuk memiliki kerangka eksperimen yang terdefinisi dengan baik. Kerangka ini harus mencakup:

- Tujuan dan hipotesis yang jelas untuk setiap eksperimen
- Metrik dan mekanisme pelacakan yang tepat untuk mengukur kinerja konten
- Strategi segmentasi dan penargetan untuk memastikan variasi konten yang relevan disampaikan ke pengguna yang tepat
- Alat analisis dan pelaporan untuk mendapatkan wawasan dari data eksperimen
- Proses untuk memasukkan pembelajaran dan optimisasi ke dalam strategi konten Anda

Dengan menerapkan iterasi dan eksperimen yang cepat, Anda dapat terus menyempurnakan dan mengoptimalkan konten Anda, memastikan bahwa konten tersebut tetap menarik, relevan, dan efektif dalam mencapai tujuan aplikasi Anda. Pendekatan tangkas dalam pembuatan konten ini memungkinkan Anda untuk tetap terdepan dan memberikan pengalaman pengguna yang luar biasa.

Skalabilitas dan Efisiensi

Seiring pertumbuhan aplikasi dan meningkatnya permintaan akan konten yang dipersonalisasi, pola pembuatan konten kontekstual memungkinkan penskalaan yang efisien

dalam pembuatan konten. LLM dapat menghasilkan konten untuk sejumlah besar pengguna dan konteks secara bersamaan, tanpa memerlukan peningkatan proporsional dalam sumber daya manusia. Skalabilitas ini memungkinkan aplikasi untuk memberikan pengalaman yang dipersonalisasi kepada basis pengguna yang berkembang tanpa membebani kemampuan pembuatan konten mereka.



Perhatikan bahwa pembuatan konten kontekstual dapat digunakan secara efektif untuk menginternasionalisasi aplikasi Anda “secara langsung”. Faktanya, itulah yang saya lakukan menggunakan Gem Instant18n saya untuk menghadirkan Olympia dalam lebih dari setengah lusin bahasa, meskipun kami belum genap berusia setahun.

Lokalisasi Bertenaga AI

Jika Anda mengizinkan saya untuk berbangga sejenak, saya pikir pustaka Instant18n saya untuk aplikasi Rails adalah contoh yang revolusioner dari pola “Pembuatan Konten Kontekstual” dalam aksi, menunjukkan potensi transformatif AI dalam pengembangan aplikasi. Gem ini memanfaatkan kekuatan model bahasa besar GPT OpenAI untuk merevolusi cara internasionalisasi dan lokalisasi ditangani dalam aplikasi Rails.

Secara tradisional, menginternasionalisasi aplikasi Rails melibatkan pendefinisian kunci terjemahan secara manual dan menyediakan terjemahan yang sesuai untuk setiap bahasa yang didukung. Proses ini bisa memakan waktu, membutuhkan banyak sumber daya, dan rentan terhadap inkonsistensi. Namun, dengan gem Instant18n, paradigma lokalisasi sepenuhnya didefinisikan ulang.

Dengan mengintegrasikan model bahasa besar, gem Instant18n memungkinkan Anda untuk menghasilkan terjemahan secara langsung, berdasarkan konteks dan makna teks. Alih-alih mengandalkan kunci terjemahan yang telah ditentukan sebelumnya dan terjemahan statis, gem ini secara dinamis menerjemahkan teks menggunakan kekuatan AI. Pendekatan ini menawarkan beberapa manfaat utama:

1. **Lokalisasi Mulus:** Dengan gem Instant18n, pengembang tidak perlu lagi mendefinisikan dan memelihara berkas terjemahan secara manual untuk setiap bahasa yang didukung. Gem ini secara otomatis menghasilkan terjemahan berdasarkan teks yang disediakan dan bahasa target yang diinginkan, membuat proses lokalisasi menjadi mudah dan mulus.
2. **Akurasi Kontekstual:** AI dapat diberikan konteks yang cukup untuk memahami nuansa teks yang diterjemahkan. AI dapat mempertimbangkan konteks sekitar, idiom, dan referensi budaya untuk menghasilkan terjemahan yang akurat, terdengar alami, dan sesuai konteks.
3. **Dukungan Bahasa yang Luas:** Gem Instant18n memanfaatkan pengetahuan luas dan kemampuan linguistik GPT, memungkinkan terjemahan ke berbagai bahasa. Mulai dari bahasa umum seperti Spanyol dan Prancis hingga bahasa yang lebih langka atau fiksi seperti Klingon dan Peri, gem ini dapat menangani berbagai kebutuhan terjemahan.
4. **Fleksibilitas dan Kreativitas:** Gem ini melampaui terjemahan bahasa tradisional dan memungkinkan opsi lokalisasi yang kreatif dan tidak konvensional. Pengembang dapat menerjemahkan teks ke berbagai gaya, dialek, atau bahkan bahasa fiksi, membuka kemungkinan baru untuk pengalaman pengguna yang unik dan konten yang menarik.
5. **Optimasi Kinerja:** Gem Instant18n menggabungkan mekanisme penyimpanan sementara untuk meningkatkan kinerja dan mengurangi beban terjemahan berulang. Teks yang diterjemahkan disimpan sementara, memungkinkan permintaan berikutnya untuk terjemahan yang sama dapat dilayani dengan cepat tanpa perlu panggilan API yang berulang.

Gem Instant18n mencontohkan kekuatan pola “Pembuatan Konten Kontekstual” dengan memanfaatkan AI untuk menghasilkan konten terlokalisasi secara dinamis. Ini menunjukkan bagaimana AI dapat diintegrasikan ke dalam fungsi inti aplikasi Rails, mengubah cara pengembang mendekati internasionalisasi dan lokalisasi.

Dengan menghilangkan kebutuhan untuk pengelolaan terjemahan manual dan memungkinkan terjemahan secara langsung berdasarkan konteks, gem Instant18n menghemat waktu dan usaha pengembang yang signifikan. Ini memungkinkan mereka untuk fokus pada membangun fitur inti aplikasi mereka sambil memastikan bahwa aspek lokalisasi ditangani dengan mulus dan akurat.

Pentingnya Pengujian Pengguna dan Umpan Balik

Akhirnya, selalu ingat pentingnya pengujian pengguna dan umpan balik. Sangat penting untuk memvalidasi bahwa pembuatan konten kontekstual memenuhi harapan pengguna dan selaras dengan tujuan aplikasi. Terus lakukan iterasi dan perbaiki konten yang dihasilkan berdasarkan wawasan pengguna dan analitik. Jika Anda menghasilkan konten dinamis dalam skala besar yang tidak mungkin divalidasi secara manual oleh Anda dan tim Anda, pertimbangkan untuk menambahkan mekanisme umpan balik yang memungkinkan pengguna melaporkan konten yang aneh atau salah, beserta penjelasan mengapa. Umpan balik berharga tersebut bahkan dapat diumpankan ke pekerja AI yang ditugaskan untuk membuat penyesuaian pada komponen yang menghasilkan konten!

Generative UI



Perhatian sangat berharga saat ini sehingga keterlibatan pengguna yang efektif kini menuntut pengalaman perangkat lunak yang tidak hanya mulus dan intuitif tetapi juga sangat dipersonalisasi sesuai kebutuhan, preferensi, dan konteks individu. Akibatnya, para desainer dan pengembang semakin menghadapi tantangan untuk menciptakan antarmuka pengguna yang dapat beradaptasi dan memenuhi kebutuhan unik setiap pengguna *dalam skala besar*.

Generative UI (GenUI) adalah pendekatan yang benar-benar revolusioner dalam desain antarmuka pengguna yang memanfaatkan kekuatan model bahasa besar (LLM) untuk menciptakan pengalaman pengguna yang sangat dipersonalisasi dan dinamis secara langsung. Saya ingin memastikan untuk setidaknya memberikan pengantar tentang GenUI dalam buku ini, karena saya percaya bahwa ini adalah salah satu peluang bidang baru yang paling menjanjikan yang saat ini ada dalam ranah desain aplikasi

dan kerangka kerja. Saya yakin bahwa puluhan atau lebih proyek komersial dan open-source baru yang sukses akan muncul dalam ceruk khusus ini.

Pada intinya, GenUI menggabungkan prinsip-prinsip [Pembuatan Konten Kontekstual](#) dengan teknik AI canggih untuk menghasilkan elemen antarmuka pengguna, seperti teks, gambar, dan tata letak, secara dinamis berdasarkan pemahaman mendalam tentang konteks, preferensi, dan tujuan pengguna. GenUI memungkinkan desainer dan pengembang untuk menciptakan antarmuka yang beradaptasi dan berkembang sebagai respons terhadap interaksi pengguna, memberikan tingkat personalisasi yang sebelumnya tidak dapat dicapai.

GenUI merepresentasikan perubahan fundamental dalam cara kita mendekati desain antarmuka pengguna. Alih-alih mendesain untuk massa, GenUI memungkinkan kita untuk mendesain untuk individu. Konten dan antarmuka yang dipersonalisasi memiliki potensi menciptakan pengalaman pengguna yang beresonansi dengan setiap pengguna pada tingkat yang lebih dalam, meningkatkan keterlibatan, kepuasan, dan loyalitas.

Sebagai teknik yang sangat mutakhir, transisi ke GenUI penuh dengan tantangan konseptual dan praktis. Mengintegrasikan AI ke dalam proses desain, memastikan bahwa antarmuka yang dihasilkan tidak hanya dipersonalisasi tetapi juga dapat digunakan, dapat diakses, dan selaras dengan merek dan pengalaman pengguna secara keseluruhan, semua ini adalah tantangan yang membuat GenUI menjadi pencarian untuk yang sedikit, bukan yang banyak. Selain itu, keterlibatan AI memunculkan pertanyaan tentang privasi data, transparansi, dan bahkan implikasi etis

Terlepas dari berbagai tantangan, pengalaman yang dipersonalisasi dalam skala besar memiliki kemampuan untuk mengubah secara total cara kita berinteraksi dengan produk dan layanan digital. Hal ini membuka kemungkinan untuk menciptakan antarmuka yang inklusif dan mudah diakses yang memenuhi beragam kebutuhan pengguna, terlepas dari kemampuan, latar belakang, atau preferensi mereka.

Dalam bab ini, kita akan mengeksplorasi konsep GenUI, memeriksa beberapa karakteristik yang menentukan, manfaat utama, dan tantangan potensial. Kita mulai

dengan mempertimbangkan bentuk GenUI yang paling mendasar dan mudah diakses: menghasilkan teks untuk antarmuka pengguna yang dirancang dan diimplementasikan secara tradisional.

Menghasilkan Teks untuk Antarmuka Pengguna

Elemen teks yang ada dalam chrome aplikasi Anda, seperti label formulir, tooltips, dan teks penjelasan, biasanya dikodekan secara langsung ke dalam template atau komponen UI, memberikan pengalaman yang konsisten tetapi generik untuk semua pengguna. Dengan menggunakan pola pembuatan konten kontekstual, Anda dapat mengubah elemen statis ini menjadi komponen yang dinamis, peka konteks, dan dipersonalisasi.

Formulir yang Dipersonalisasi

Formulir adalah bagian yang ada di mana-mana dalam aplikasi web dan seluler, berfungsi sebagai sarana utama untuk mengumpulkan input pengguna. Namun, formulir tradisional seringkali menyajikan pengalaman yang generik dan tidak personal, dengan label dan bidang standar yang mungkin tidak selalu sesuai dengan konteks atau kebutuhan spesifik pengguna. Pengguna lebih cenderung mengisi formulir yang terasa disesuaikan dengan kebutuhan dan preferensi mereka, yang mengarah pada tingkat konversi dan kepuasan pengguna yang lebih tinggi.

Namun, penting untuk menjaga keseimbangan antara personalisasi dan konsistensi. Meskipun menyesuaikan formulir untuk pengguna individual bisa bermanfaat, sangat penting untuk mempertahankan tingkat keakraban dan prediktabilitas. Pengguna tetap harus dapat mengenali dan menavigasi formulir dengan mudah, bahkan dengan elemen yang dipersonalisasi.

Berikut beberapa ide formulir yang dipersonalisasi untuk inspirasi:

Saran Isian Kontekstual

GenUI dapat menganalisis interaksi sebelumnya, preferensi, dan data pengguna untuk memberikan saran isian yang cerdas sebagai prediksi. Misalnya, jika pengguna sebelumnya telah memasukkan alamat pengiriman mereka, formulir dapat secara otomatis mengisi bidang yang relevan dengan informasi yang tersimpan. Ini tidak hanya menghemat waktu tetapi juga menunjukkan bahwa aplikasi memahami dan mengingat preferensi pengguna.

Tunggu sebentar, bukankah teknik ini bisa dilakukan tanpa melibatkan AI? Tentu saja, tetapi keindahan menggerakkan fungsionalitas semacam ini dengan AI ada dua hal: 1) betapa mudahnya untuk diimplementasikan dan 2) betapa tangguhnyanya saat UI Anda berubah dan berkembang seiring waktu.

Mari kita buat layanan untuk sistem penanganan pesanan teoretis kita, yang mencoba secara proaktif mengisi alamat pengiriman yang tepat untuk pengguna.

```
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
```

Contoh ini sangat disederhanakan, tetapi seharusnya bisa diterapkan untuk kebanyakan kasus. Idenya adalah membiarkan AI membuat perkiraan seperti yang akan dilakukan manusia. Untuk memperjelas apa yang saya maksud, mari kita lihat beberapa contoh data:

```
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]

Apakah Anda menyadari pola dalam data tersebut? Saya jamin ini adalah hal yang mudah bagi LLM. Untuk membuktikannya, mari kita tanyakan kepada GPT-4 alamat pengiriman mana yang paling mungkin untuk sebuah “termometer”.

```
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.
```

Jika Anda berpikir bahwa menggunakan model mahal seperti GPT-4 untuk tugas ini terlalu berlebihan, Anda benar! Saya mencoba prompt yang sama pada Mistral 7B Instruct dan menghasilkan respons berikut dengan kecepatan 75 token per detik, dengan biaya yang sangat rendah yaitu \$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.

Apakah beban dan biaya dari teknik ini sepadan untuk membuat pengalaman checkout lebih ajaib? Bagi banyak peritel daring, jawabannya tentu saja ya. Dan dari yang terlihat, biaya komputasi AI hanya akan semakin turun, terutama untuk penyedia hosting model open source yang bersaing ketat untuk menawarkan harga terendah.



Gunakan [Prompt Template](#) dan [StructuredIO](#) bersama dengan [Response Fencing](#) untuk mengoptimalkan jenis chat completion seperti ini.

Pengurutan Kolom Adaptif

Urutan penyajian kolom formulir dapat berdampak signifikan pada pengalaman pengguna dan tingkat penyelesaian. Dengan GenUI, Anda dapat menyesuaikan urutan kolom secara dinamis berdasarkan konteks pengguna dan tingkat kepentingan setiap kolom. Misalnya, jika pengguna sedang mengisi formulir pendaftaran untuk aplikasi kebugaran, formulir tersebut bisa memprioritaskan kolom-kolom yang terkait dengan tujuan dan preferensi kebugaran mereka, membuat prosesnya lebih relevan dan menarik.

Personalized Microcopy

Teks instruksi, pesan kesalahan, dan microcopy lain yang terkait dengan formulir juga dapat dipersonalisasi menggunakan GenUI. Alih-alih menampilkan pesan kesalahan generik seperti “Alamat email tidak valid,” Anda dapat menghasilkan pesan yang lebih membantu dan kontekstual seperti “Harap masukkan alamat email yang valid untuk menerima konfirmasi pesanan Anda.” Sentuhan personal ini dapat membuat pengalaman mengisi formulir lebih ramah pengguna dan tidak membuat frustrasi.

Validasi Personal

Sejalan dengan Personalized Microcopy, Anda dapat menggunakan AI untuk memvalidasi formulir dengan cara yang terasa ajaib. Bayangkan membiarkan AI memvalidasi formulir profil pengguna, mencari kemungkinan kesalahan pada tingkat *semantik*.

Create your account

Full name

Obie Fernandez

Email

obiefernandez@gmail.com



Did you mean obiefernandez@gmail.com? [Yes, update.](#)

Country ⓘ

 United States



Password

.....



✔ Nice work. This is an excellent password.

Gambar 9. Bisakah Anda melihat validasi semantik yang sedang berlangsung?

Pengungkapan Progresif

GenUI dapat secara cerdas menentukan bidang formulir mana yang penting berdasarkan konteks pengguna dan secara bertahap menampilkan bidang tambahan sesuai kebutuhan. Teknik pengungkapan progresif ini membantu mengurangi beban kognitif dan membuat proses pengisian formulir lebih mudah dikelola. Misalnya, jika

pengguna mendaftar untuk langganan dasar, formulir awalnya hanya menampilkan bidang-bidang yang penting, dan ketika pengguna melanjutkan atau memilih opsi tertentu, bidang tambahan yang relevan dapat dimunculkan secara dinamis.

Teks Penjelasan Berbasis Konteks

Tooltips sering digunakan untuk memberikan informasi tambahan atau panduan kepada pengguna ketika mereka mengarahkan kursor atau berinteraksi dengan elemen tertentu. Dengan pendekatan “Pembuatan Konten Kontekstual”, Anda dapat menghasilkan tooltips yang beradaptasi dengan konteks pengguna dan memberikan informasi yang relevan. Misalnya, jika pengguna sedang menjelajahi fitur yang kompleks, tooltip dapat menawarkan tips atau contoh yang dipersonalisasi berdasarkan interaksi sebelumnya atau tingkat kemampuan mereka.

Teks penjelasan, seperti instruksi, deskripsi, atau pesan bantuan, dapat dihasilkan secara dinamis berdasarkan konteks pengguna. Alih-alih menyajikan penjelasan umum, Anda dapat menggunakan LLM untuk menghasilkan teks yang disesuaikan dengan kebutuhan atau pertanyaan spesifik pengguna. Sebagai contoh, jika pengguna mengalami kesulitan dengan langkah tertentu dalam suatu proses, teks penjelasan dapat memberikan panduan atau tips pemecahan masalah yang dipersonalisasi.

Microcopy mengacu pada potongan-potongan teks kecil yang memandu pengguna melalui aplikasi Anda, seperti label tombol, pesan kesalahan, atau prompt konfirmasi. Dengan menerapkan pendekatan [Pembuatan Konten Kontekstual](#) pada microcopy, Anda dapat menciptakan UI yang adaptif yang merespons tindakan pengguna dan memberikan teks yang relevan dan membantu. Misalnya, jika pengguna akan melakukan tindakan penting, prompt konfirmasi dapat dihasilkan secara dinamis untuk memberikan pesan yang jelas dan personal.

Teks penjelasan dan tooltips yang dipersonalisasi dapat sangat meningkatkan proses onboarding untuk pengguna baru. Dengan memberikan panduan dan contoh yang

spesifik sesuai konteks, Anda dapat membantu pengguna dengan cepat memahami dan menavigasi aplikasi, mengurangi kurva pembelajaran dan meningkatkan adopsi.

Elemen chrome yang dinamis dan peka konteks juga dapat membuat aplikasi terasa lebih intuitif dan menarik. Pengguna lebih cenderung berinteraksi dengan dan mengeksplorasi fitur-fitur ketika teks yang menyertainya disesuaikan dengan kebutuhan dan minat spesifik mereka.

Sejauh ini kita telah membahas ide-ide untuk meningkatkan paradigma UI yang ada dengan AI, tetapi bagaimana dengan memikirkan kembali cara antarmuka pengguna dirancang dan diimplementasikan dengan cara yang lebih radikal?

Mendefinisikan UI Generatif

Berbeda dengan desain UI tradisional, di mana desainer membuat antarmuka tetap dan statis, GenUI mengisyaratkan masa depan di mana perangkat lunak kita memiliki pengalaman yang fleksibel dan personal yang dapat berkembang dan beradaptasi secara real-time. Setiap kali kita menggunakan antarmuka percakapan berbasis AI, kita membiarkan AI beradaptasi dengan kebutuhan khusus pengguna. GenUI melangkah lebih jauh dengan menerapkan tingkat adaptabilitas tersebut pada antarmuka visual perangkat lunak.

Alasan mengapa kita dapat bereksperimen dengan ide-ide GenUI saat ini adalah karena model bahasa besar sudah memahami pemrograman dan pengetahuan dasarnya mencakup teknologi dan kerangka kerja UI. Pertanyaannya adalah apakah model bahasa besar dapat digunakan untuk menghasilkan elemen UI, seperti teks, gambar, tata letak, dan bahkan antarmuka lengkap, yang disesuaikan dengan setiap pengguna individual. Model tersebut dapat diinstruksikan untuk mempertimbangkan berbagai faktor, seperti interaksi pengguna sebelumnya, preferensi yang dinyatakan, informasi demografis, dan

konteks penggunaan saat ini, untuk menciptakan antarmuka yang sangat personal dan relevan.

GenUI berbeda dari desain antarmuka pengguna tradisional dalam beberapa aspek utama:

1. **Dinamis dan Adaptif:** Desain UI tradisional melibatkan pembuatan antarmuka tetap dan statis yang tetap sama untuk semua pengguna. Sebaliknya, GenUI memungkinkan antarmuka yang dapat beradaptasi dan berubah secara dinamis berdasarkan kebutuhan dan konteks pengguna. Ini berarti aplikasi yang sama dapat menampilkan antarmuka yang berbeda untuk pengguna yang berbeda atau bahkan untuk pengguna yang sama dalam situasi yang berbeda.
2. **Personalisasi Skala Besar:** Dengan desain tradisional, membuat pengalaman yang dipersonalisasi untuk setiap pengguna seringkali tidak praktis karena waktu dan sumber daya yang dibutuhkan. Sebaliknya, GenUI memungkinkan personalisasi dalam skala besar. Dengan memanfaatkan AI, desainer dapat membuat antarmuka yang secara otomatis beradaptasi dengan kebutuhan dan preferensi unik setiap pengguna, tanpa harus merancang dan mengembangkan antarmuka terpisah untuk setiap segmen pengguna.
3. **Fokus pada Hasil:** Desain UI tradisional sering berfokus pada pembuatan antarmuka yang menarik secara visual dan fungsional. Meskipun aspek-aspek ini masih penting dalam GenUI, fokus utama beralih ke pencapaian hasil yang diinginkan pengguna. GenUI bertujuan untuk menciptakan antarmuka yang dioptimalkan untuk tujuan dan tugas spesifik setiap pengguna, memprioritaskan kegunaan dan efektivitas di atas pertimbangan estetika semata.
4. **Pembelajaran dan Peningkatan Berkelanjutan:** Sistem GenUI dapat terus belajar dan meningkat seiring waktu berdasarkan interaksi dan umpan balik pengguna. Saat pengguna berinteraksi dengan antarmuka yang dihasilkan, model AI dapat mengumpulkan data tentang perilaku, preferensi, dan hasil pengguna, menggunakan informasi ini untuk menyempurnakan dan mengoptimalkan generasi antarmuka di masa depan. Proses pembelajaran iteratif ini memungkinkan

sistem GenUI menjadi semakin efektif dalam memenuhi kebutuhan pengguna seiring waktu.

Penting untuk dicatat bahwa GenUI berbeda dengan alat desain berbantuan AI, seperti yang memberikan saran atau mengotomatisasi tugas-tugas desain tertentu. Meskipun alat-alat ini dapat membantu memperlancar proses desain, mereka masih bergantung pada desainer untuk membuat keputusan akhir dan menciptakan antarmuka statis. GenUI, di sisi lain, melibatkan sistem AI yang mengambil peran lebih aktif dalam pembuatan dan adaptasi antarmuka berdasarkan data dan konteks pengguna.

GenUI merepresentasikan pergeseran signifikan dalam cara kita mendekati desain antarmuka pengguna, bergerak dari solusi satu-ukuran-untuk-semua menuju pengalaman yang sangat personal dan adaptif. Dengan memanfaatkan kekuatan AI, GenUI memiliki potensi untuk merevolusi cara kita berinteraksi dengan produk dan layanan digital, menciptakan antarmuka yang lebih intuitif, menarik, dan efektif untuk setiap pengguna individual.

Contoh

Untuk mengilustrasikan konsep GenUI, mari kita pertimbangkan sebuah aplikasi kebugaran hipotetis bernama “FitAI”. Aplikasi ini bertujuan untuk menyediakan rencana latihan dan saran nutrisi yang dipersonalisasi untuk pengguna berdasarkan tujuan, tingkat kebugaran, dan preferensi individual mereka.

Dalam pendekatan desain UI tradisional, FitAI mungkin memiliki serangkaian layar dan elemen tetap yang sama untuk semua pengguna. Namun, dengan GenUI, antarmuka aplikasi dapat beradaptasi secara dinamis dengan kebutuhan dan konteks unik setiap pengguna.

Pendekatan ini agak sulit dibayangkan untuk diimplementasikan pada tahun 2024 dan mungkin bahkan tidak memiliki ROI yang memadai, tetapi hal ini memungkinkan untuk dilakukan.

Berikut cara kerjanya:

1. Proses Pengenalan:

- Alih-alih kuesioner standar, FitAI menggunakan AI konversasional untuk mengumpulkan informasi tentang tujuan, tingkat kebugaran saat ini, dan preferensi pengguna.
- Berdasarkan interaksi awal ini, AI menghasilkan tata letak dasbor yang dipersonalisasi, menyoroti fitur dan informasi yang paling relevan dengan tujuan pengguna.
- Teknologi AI saat ini mungkin memiliki pilihan komponen layar yang dapat digunakan dalam menyusun dasbor yang dipersonalisasi.
- Teknologi AI masa depan mungkin mengambil peran desainer UI berpengalaman dan benar-benar menciptakan dasbor *dari awal*.

2. Perencana Latihan:

- Antarmuka perencana latihan diadaptasi oleh AI berdasarkan tingkat pengalaman pengguna dan peralatan yang tersedia.
- Untuk pemula tanpa peralatan, mungkin menunjukkan latihan sederhana menggunakan berat badan dengan instruksi dan video yang detail.
- Untuk pengguna tingkat lanjut dengan akses ke gym, bisa menampilkan rutinitas yang lebih kompleks dengan konten penjelasan yang lebih sedikit.
- Konten perencana latihan tidak hanya difilter dari superset besar. Konten dapat dihasilkan *secara langsung* berdasarkan basis pengetahuan yang diquery dengan konteks yang mencakup semua hal yang diketahui tentang pengguna.

3. Pelacakan Kemajuan:

- Antarmuka pelacakan kemajuan berkembang berdasarkan tujuan dan pola keterlibatan pengguna.

- Jika pengguna terutama berfokus pada penurunan berat badan, antarmuka mungkin akan menampilkan grafik tren berat badan dan statistik pembakaran kalori secara menonjol.
- Untuk pengguna yang sedang membangun otot, antarmuka bisa menonjolkan peningkatan kekuatan dan perubahan komposisi tubuh.
- AI dapat menyesuaikan bagian aplikasi ini dengan kemajuan aktual pengguna. Jika kemajuan terhenti untuk periode waktu tertentu, aplikasi dapat beralih ke mode di mana ia mencoba membujuk pengguna untuk mengungkapkan alasan kemunduran tersebut, untuk mengatasi masalah tersebut.

4. Saran Nutrisi:

- Bagian nutrisi menyesuaikan dengan preferensi dan pembatasan makanan pengguna.
- Untuk pengguna vegan, bagian ini mungkin menunjukkan saran makanan nabati dan sumber protein.
- Untuk pengguna dengan intoleransi gluten, bagian ini akan secara otomatis menyaring makanan yang mengandung gluten dari rekomendasi.
- Sekali lagi, konten tidak diambil dari kumpulan data makanan yang sangat besar yang berlaku untuk semua pengguna, melainkan disintesis dari basis pengetahuan yang berisi informasi yang dapat disesuaikan berdasarkan situasi dan batasan spesifik pengguna.
- Misalnya, resep dibuat dengan spesifikasi bahan yang sesuai dengan kebutuhan kalori pengguna yang terus berubah seiring dengan evolusi tingkat kebugaran dan statistik tubuh mereka.

5. Elemen Motivasi:

- Konten motivasi dan notifikasi aplikasi dipersonalisasi berdasarkan tipe kepribadian pengguna dan respons terhadap berbagai strategi motivasi.

- Beberapa pengguna mungkin menerima pesan yang mendorong semangat, sementara yang lain mendapatkan umpan balik yang lebih berbasis data.

Dalam contoh ini, GenUI memungkinkan FitAI untuk menciptakan pengalaman yang sangat disesuaikan untuk setiap pengguna, berpotensi meningkatkan keterlibatan, kepuasan, dan kemungkinan mencapai tujuan kebugaran. Elemen antarmuka, konten, dan bahkan “kepribadian” aplikasi beradaptasi untuk melayani kebutuhan dan preferensi setiap pengguna individual dengan sebaik-baiknya.

Peralihan ke Desain Berorientasi Hasil

GenUI mewakili pergeseran fundamental dalam pendekatan desain antarmuka pengguna!, beralih dari fokus pada pembuatan elemen antarmuka spesifik ke pendekatan yang lebih holistik dan berorientasi hasil. Pergeseran ini memiliki beberapa implikasi penting:

1. Fokus pada Tujuan Pengguna:

- Desainer perlu berpikir lebih dalam tentang tujuan pengguna dan hasil yang diinginkan daripada komponen antarmuka spesifik.
- Penekanannya akan pada pembuatan sistem yang dapat menghasilkan antarmuka yang membantu pengguna mencapai tujuan mereka secara efisien dan efektif.
- Kerangka kerja UI baru akan muncul yang memberi perancang berbasis AI alat yang mereka butuhkan untuk dapat menghasilkan pengalaman pengguna *secara langsung* dan *dari awal* alih-alih berdasarkan spesifikasi layar yang telah ditentukan sebelumnya.

2. Peran Desainer yang Berubah:

- Desainer akan bertransisi dari membuat tata letak tetap menjadi mendefinisikan aturan, batasan, dan pedoman yang harus diikuti sistem AI ketika menghasilkan antarmuka.
- Mereka perlu mengembangkan keterampilan di bidang seperti analisis data, prompt engineering, dan pemikiran sistemik untuk secara efektif mengarahkan sistem GenUI.

3. Pentingnya Riset Pengguna:

- Riset pengguna menjadi semakin penting dalam konteks GenUI, karena desainer perlu memahami tidak hanya preferensi pengguna, tetapi juga bagaimana preferensi dan kebutuhan ini berubah dalam konteks yang berbeda.
- Pengujian pengguna yang berkelanjutan dan putaran umpan balik akan sangat penting untuk menyempurnakan dan meningkatkan kemampuan AI dalam menghasilkan antarmuka yang efektif.

4. Merancang untuk Variabilitas:

- Alih-alih menciptakan satu antarmuka “sempurna”, desainer perlu mempertimbangkan berbagai variasi yang mungkin dan memastikan bahwa sistem dapat menghasilkan antarmuka yang sesuai untuk berbagai kebutuhan pengguna.
- Ini termasuk merancang untuk kasus-kasus ekstrem dan memastikan bahwa antarmuka yang dihasilkan mempertahankan kegunaan dan aksesibilitas di berbagai konfigurasi.
- Diferensiasi produk mengambil dimensi baru yang melibatkan perspektif berbeda tentang psikologi pengguna dan pemanfaatan kumpulan data serta basis pengetahuan unik yang tidak tersedia bagi pesaing.

Tantangan dan Pertimbangan

Sementara GenUI menawarkan kemungkinan yang menarik, ia juga menghadirkan beberapa tantangan dan pertimbangan:

1. Keterbatasan Teknis:

- Teknologi AI saat ini, meski sudah maju, masih memiliki keterbatasan dalam memahami maksud pengguna yang kompleks dan menghasilkan antarmuka yang benar-benar sadar konteks.
- Masalah kinerja terkait pembuatan elemen antarmuka secara real-time, terutama pada perangkat dengan daya komputasi lebih rendah.

2. Persyaratan Data:

- Tergantung pada kasus penggunaan, sistem GenUI yang efektif mungkin memerlukan jumlah data pengguna yang signifikan untuk menghasilkan antarmuka yang dipersonalisasi.
- Tantangan dalam memperoleh data pengguna secara etis menimbulkan kekhawatiran tentang privasi dan keamanan data, serta potensi bias dalam data yang digunakan untuk melatih model GenUI.

3. Kegunaan dan Konsistensi:

- Setidaknya sampai praktik ini menjadi lazim, aplikasi dengan antarmuka yang terus berubah dapat menyebabkan masalah kegunaan, karena pengguna mungkin kesulitan menemukan elemen yang familiar atau bernavigasi secara efisien.
- Mencapai keseimbangan antara personalisasi dan mempertahankan antarmuka yang konsisten dan dapat dipelajari akan sangat penting.

4. Ketergantungan Berlebihan pada AI:

- Ada risiko terlalu mendelegasikan keputusan desain kepada sistem AI, yang berpotensi menghasilkan pilihan antarmuka yang tidak inspiratif, bermasalah, atau rusak.
- Pengawasan manusia dan kemampuan untuk mengesampingkan desain yang dihasilkan AI akan tetap penting dalam masa mendatang.

5. Pertimbangan Aksesibilitas:

- Memastikan antarmuka yang dihasilkan secara dinamis tetap dapat diakses oleh pengguna dengan disabilitas menghadirkan tantangan baru yang mengkhawatirkan, mengingat tingkat kepatuhan aksesibilitas yang buruk yang ditunjukkan oleh sistem pada umumnya.
- Di sisi lain, perancang AI mungkin diimplementasikan dengan kepedulian terhadap aksesibilitas yang sudah *tertanam*, dan kemampuan untuk membangun antarmuka yang dapat diakses secara langsung seperti mereka membangun UI untuk pengguna tanpa hambatan.
- Bagaimanapun juga, sistem GenUI harus dirancang dengan pedoman aksesibilitas dan proses pengujian yang kuat.

6. Kepercayaan dan Transparansi Pengguna:

- Pengguna mungkin merasa tidak nyaman dengan antarmuka yang tampak “terlalu mengenal” mereka atau berubah dengan cara yang tidak mereka pahami.
- Memberikan transparansi tentang bagaimana dan mengapa antarmuka dipersonalisasi akan penting untuk membangun kepercayaan pengguna.

Prospek dan Peluang Masa Depan

Masa depan UI Generatif (GenUI) menyimpan janji besar untuk merevolusi cara kita berinteraksi dengan produk dan layanan digital. Seiring teknologi ini terus berkembang,

kita dapat mengantisipasi pergeseran besar dalam cara antarmuka pengguna dirancang, diimplementasikan, dan dialami. Saya pikir GenUI adalah fenomena yang akhirnya akan mendorong perangkat lunak kita ke dalam ranah yang saat ini dianggap sebagai fiksi ilmiah.

Salah satu prospek paling menarik dari GenUI adalah potensinya untuk meningkatkan aksesibilitas pada skala besar yang melampaui sekadar memastikan bahwa orang dengan disabilitas serius tidak sepenuhnya dikecualikan dari penggunaan perangkat lunak Anda. Dengan secara otomatis menyesuaikan antarmuka dengan kebutuhan individual pengguna, GenUI dapat membuat pengalaman digital lebih inklusif dari sebelumnya. Bayangkan antarmuka yang dengan mulus menyesuaikan untuk menyediakan teks yang lebih besar bagi pengguna yang lebih muda atau yang memiliki gangguan penglihatan, atau tata letak yang disederhanakan bagi mereka yang memiliki keterbatasan kognitif, semuanya tanpa memerlukan konfigurasi manual atau versi aplikasi “yang dapat diakses” secara terpisah.

Kemampuan personalisasi GenUI kemungkinan akan mendorong peningkatan keterlibatan, kepuasan, dan loyalitas pengguna di berbagai produk digital. Ketika antarmuka menjadi lebih selaras dengan preferensi dan perilaku individual, pengguna akan menemukan pengalaman digital yang lebih intuitif dan menyenangkan, yang berpotensi mengarah pada interaksi yang lebih dalam dan lebih bermakna dengan teknologi.

GenUI juga memiliki potensi untuk mengubah proses pengenalan bagi pengguna baru. Dengan menciptakan pengalaman pengguna pertama kali yang intuitif dan dipersonalisasi yang dengan cepat beradaptasi dengan tingkat keahlian masing-masing pengguna, GenUI dapat secara signifikan mengurangi kurva pembelajaran yang terkait dengan aplikasi baru. Hal ini dapat mengarah pada tingkat adopsi yang lebih cepat dan peningkatan kepercayaan diri pengguna dalam mengeksplorasi fitur dan fungsionalitas baru.

Kemungkinan menarik lainnya adalah kemampuan GenUI untuk mempertahankan pengalaman pengguna yang konsisten di berbagai perangkat dan platform sambil

mengoptimalkan untuk setiap konteks penggunaan tertentu. Ini bisa menyelesaikan tantangan lama dalam menyediakan pengalaman yang koheren di berbagai perangkat yang semakin beragam, mulai dari ponsel pintar dan tablet hingga komputer desktop dan teknologi yang sedang berkembang seperti kacamata realitas ditambah

Sifat GenUI yang berbasis data membuka peluang untuk iterasi dan peningkatan yang cepat dalam desain antarmuka pengguna. Dengan mengumpulkan data real-time tentang bagaimana pengguna berinteraksi dengan antarmuka yang dihasilkan, desainer dan pengembang dapat memperoleh wawasan yang belum pernah ada sebelumnya tentang perilaku dan preferensi pengguna. Siklus umpan balik ini dapat menghasilkan peningkatan berkelanjutan dalam desain antarmuka pengguna, yang didorong oleh pola penggunaan aktual daripada asumsi atau pengujian pengguna yang terbatas.

Untuk mempersiapkan perubahan ini, desainer perlu mengembangkan keterampilan dan pola pikir mereka. Fokus akan beralih dari membuat tata letak tetap menjadi mengembangkan sistem desain dan pedoman yang komprehensif yang dapat menginformasikan pembuatan antarmuka berbasis AI. Desainer perlu mengembangkan pemahaman mendalam tentang analisis data, teknologi AI, dan pemikiran sistemis untuk secara efektif memandu sistem GenUI.

Selain itu, karena GenUI mengaburkan batas antara desain dan teknologi, desainer perlu berkolaborasi lebih erat dengan pengembang dan ilmuwan data. Pendekatan interdisipliner ini akan sangat penting dalam menciptakan sistem GenUI yang tidak hanya menarik secara visual dan ramah pengguna, tetapi juga kuat secara teknis dan etis.

Implikasi etis dari GenUI juga akan menjadi hal utama seiring dengan berkembangnya teknologi. Desainer akan memainkan peran penting dalam mengembangkan kerangka kerja untuk penggunaan AI yang bertanggung jawab dalam desain antarmuka, memastikan bahwa personalisasi meningkatkan pengalaman pengguna tanpa mengorbankan privasi atau memanipulasi perilaku pengguna dengan cara yang tidak etis.

Ketika kita melihat ke masa depan, GenUI menghadirkan peluang yang menarik sekali-

gus tantangan yang signifikan. Ini memiliki potensi untuk menciptakan pengalaman digital yang lebih intuitif, efisien, dan memuaskan bagi pengguna di seluruh dunia. Meskipun akan membutuhkan desainer untuk beradaptasi dan memperoleh keterampilan baru, ini juga menawarkan kesempatan yang belum pernah ada sebelumnya untuk membentuk masa depan interaksi manusia-komputer dengan cara yang mendalam dan bermakna. Perjalanan menuju sistem GenUI yang sepenuhnya terealisasi tentunya akan kompleks, tetapi potensi manfaat dalam hal peningkatan pengalaman pengguna dan aksesibilitas digital menjadikannya masa depan yang layak diperjuangkan.

Orkestrasi Alur Kerja Cerdas



Dalam ranah pengembangan aplikasi, alur kerja memainkan peran penting dalam mendefinisikan bagaimana tugas, proses, dan interaksi pengguna disusun dan dijalankan. Seiring aplikasi menjadi semakin kompleks dan ekspektasi pengguna terus meningkat, kebutuhan akan orkestrasi alur kerja yang cerdas dan adaptif menjadi semakin nyata.

Pendekatan “Orkestrasi Alur Kerja Cerdas” berfokus pada pemanfaatan komponen AI untuk mengorkestrasi dan mengoptimalkan alur kerja kompleks dalam aplikasi secara dinamis. Tujuannya adalah untuk menciptakan aplikasi yang lebih efisien, responsif, dan dapat beradaptasi dengan data dan konteks secara real-time.

Dalam bab ini, kita akan mengeksplorasi prinsip-prinsip utama dan pola yang mendasari pendekatan orkestrasi alur kerja cerdas. Kita akan mempertimbangkan bagaimana AI dapat digunakan untuk mengarahkan tugas secara cerdas, mengotomatisasi pengambilan keputusan, dan mengadaptasi alur kerja secara dinamis berdasarkan berbagai faktor

seperti perilaku pengguna, kinerja sistem, dan aturan bisnis. Melalui contoh praktis dan skenario dunia nyata, kita akan mendemonstrasikan potensi transformatif AI dalam merampingkan dan mengoptimalkan alur kerja aplikasi.

Baik Anda sedang membangun aplikasi enterprise dengan proses bisnis yang rumit atau aplikasi yang menghadap konsumen dengan perjalanan pengguna yang dinamis, pola dan teknik yang dibahas dalam bab ini akan membekali Anda dengan pengetahuan dan alat untuk menciptakan alur kerja yang cerdas dan efisien yang meningkatkan pengalaman pengguna secara keseluruhan dan mendorong nilai bisnis.

Kebutuhan Bisnis

Pendekatan tradisional untuk manajemen alur kerja sering bergantung pada aturan yang telah ditentukan sebelumnya dan pohon keputusan statis, yang dapat menjadi kaku, tidak fleksibel, dan tidak mampu mengatasi sifat dinamis aplikasi modern.

Pertimbangkan skenario di mana aplikasi e-commerce perlu menangani proses pemenuhan pesanan yang kompleks. Alur kerja mungkin melibatkan beberapa langkah seperti validasi pesanan, pemeriksaan inventaris, pemrosesan pembayaran, pengiriman, dan pemberitahuan pelanggan. Setiap langkah mungkin memiliki seperangkat aturan sendiri, ketergantungan, integrasi eksternal, dan mekanisme penanganan pengecualian. Mengelola alur kerja seperti itu secara manual atau melalui logika yang dikodekan secara langsung dapat dengan cepat menjadi rumit, rawan kesalahan, dan sulit untuk dipertahankan.

Selain itu, seiring aplikasi berkembang dan jumlah pengguna simultan bertambah, alur kerja mungkin perlu beradaptasi dan mengoptimalkan dirinya berdasarkan data waktu nyata dan kinerja sistem. Sebagai contoh, selama periode lalu lintas puncak, aplikasi mungkin perlu menyesuaikan alur kerja secara dinamis untuk memprioritaskan tugas-tugas tertentu, mengalokasikan sumber daya secara efisien, dan memastikan pengalaman pengguna yang lancar.

Di sinilah pendekatan “Orkestrasi Alur Kerja Cerdas” berperan. Dengan memanfaatkan komponen AI, pengembang dapat menciptakan alur kerja yang cerdas, adaptif, dan dapat mengoptimalkan diri. AI dapat menganalisis data dalam jumlah besar, belajar dari pengalaman sebelumnya, dan membuat keputusan berdasarkan informasi secara waktu nyata untuk mengorkestrasi alur kerja secara efektif.

Manfaat Utama

1. **Peningkatan Efisiensi:** AI dapat mengoptimalkan alokasi tugas, pemanfaatan sumber daya, dan eksekusi alur kerja, yang menghasilkan waktu pemrosesan lebih cepat dan efisiensi keseluruhan yang lebih baik.
2. **Kemampuan Beradaptasi:** Alur kerja berbasis AI dapat beradaptasi secara dinamis terhadap perubahan kondisi, seperti fluktuasi permintaan pengguna, kinerja sistem, atau kebutuhan bisnis, memastikan aplikasi tetap responsif dan tangguh.
3. **Pengambilan Keputusan Otomatis:** AI dapat mengotomatisasi proses pengambilan keputusan yang kompleks dalam alur kerja, mengurangi intervensi manual dan meminimalkan risiko kesalahan manusia.
4. **Personalisasi:** AI dapat menganalisis perilaku pengguna, preferensi, dan konteks untuk mempersonalisasi alur kerja dan memberikan pengalaman yang disesuaikan untuk setiap pengguna.
5. **Skalabilitas:** Alur kerja yang didukung AI dapat berkembang secara mulus untuk menangani peningkatan volume data dan interaksi pengguna, tanpa mengorbankan kinerja atau keandalan.

Dalam bagian-bagian berikut, kita akan mengeksplorasi pola dan teknik utama yang memungkinkan implementasi alur kerja cerdas dan menampilkan contoh dunia nyata tentang bagaimana AI mengubah manajemen alur kerja dalam aplikasi modern.

Pola-pola Utama

Untuk mengimplementasikan orkestrasi alur kerja cerdas dalam aplikasi, pengembang dapat memanfaatkan beberapa pola utama yang memanfaatkan kekuatan AI. Pola-pola ini memberikan pendekatan terstruktur untuk merancang dan mengelola alur kerja, memungkinkan aplikasi untuk beradaptasi, mengoptimalkan, dan mengotomatisasi proses berdasarkan data waktu nyata dan konteks. Mari kita eksplorasi beberapa pola fundamental dalam orkestrasi alur kerja cerdas.

Perutean Tugas Dinamis

Pola ini melibatkan penggunaan AI untuk merutekan tugas secara cerdas dalam alur kerja berdasarkan berbagai faktor seperti prioritas tugas, ketersediaan sumber daya, dan kinerja sistem. Algoritma AI dapat menganalisis karakteristik setiap tugas, mempertimbangkan keadaan sistem saat ini, dan membuat keputusan berdasarkan informasi untuk menetapkan tugas ke sumber daya atau jalur pemrosesan yang paling sesuai. Perutean tugas dinamis memastikan bahwa tugas-tugas didistribusikan dan dieksekusi secara efisien, mengoptimalkan kinerja alur kerja secara keseluruhan.

```
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
```

Perhatikan perulangan yang dibuat oleh ekspresi `while` pada baris 29, yang terus meminta AI sampai tugas ditetapkan. Pada baris 35, kita menyimpan transkrip tugas untuk analisis dan pencarian dan perbaikan bug di kemudian hari, jika diperlukan.

Pengambilan Keputusan Kontekstual

Anda dapat menggunakan kode yang sangat mirip untuk membuat keputusan berbasis konteks dalam alur kerja. Dengan menganalisis poin data yang relevan seperti preferensi pengguna, pola historis, dan masukan waktu nyata, komponen AI dapat menentukan tindakan yang paling tepat pada setiap titik keputusan dalam alur kerja. Sesuaikan perilaku alur kerja Anda berdasarkan konteks spesifik setiap pengguna atau skenario, memberikan pengalaman yang dipersonalisasi dan dioptimalkan.

Komposisi Alur Kerja Adaptif

Pola ini berfokus pada penyusunan dan penyesuaian alur kerja secara dinamis berdasarkan perubahan persyaratan atau kondisi. AI dapat menganalisis keadaan alur kerja saat ini, mengidentifikasi hambatan atau ketidakefisienan, dan secara otomatis memodifikasi struktur alur kerja untuk mengoptimalkan kinerja. Komposisi alur kerja adaptif memungkinkan aplikasi untuk terus berkembang dan meningkatkan prosesnya tanpa memerlukan intervensi manual.

Penanganan dan Pemulihan Pengecualian

Penanganan dan pemulihan pengecualian adalah aspek penting dalam orkestrasi alur kerja cerdas. Ketika bekerja dengan komponen AI dan alur kerja yang kompleks, penting untuk mengantisipasi dan menangani pengecualian dengan anggun untuk memastikan stabilitas dan keandalan sistem.

Berikut adalah beberapa pertimbangan dan teknik penting untuk penanganan dan pemulihan pengecualian dalam alur kerja cerdas:

1. **Propagasi Pengecualian:** Terapkan pendekatan yang konsisten untuk mempropagasi pengecualian di seluruh komponen alur kerja. Ketika pengecualian terjadi dalam suatu komponen, pengecualian tersebut harus ditangkap, dicatat, dan dipropagasi ke orkestrator atau komponen terpisah yang bertanggung jawab untuk menangani pengecualian. Idenya adalah untuk memusatkan penanganan pengecualian dan mencegah pengecualian ditelan secara diam-diam, serta membuka kemungkinan untuk [Penanganan Kesalahan Cerdas](#).
2. **Mekanisme Percobaan Ulang:** Mekanisme percobaan ulang membantu meningkatkan ketahanan alur kerja dan menangani kegagalan sementara dengan anggun. Pastikan untuk menerapkan mekanisme percobaan ulang untuk pengecualian yang bersifat sementara atau dapat dipulihkan, seperti konektivitas

jaringan atau ketidaktersediaan sumber daya yang dapat dicoba ulang secara otomatis setelah penundaan tertentu. Memiliki orkestrator atau penanganan pengecualian yang ditenagai AI berarti bahwa strategi percobaan ulang Anda tidak harus bersifat mekanis, mengandalkan algoritma tetap seperti fallback eksponensial. Anda dapat menyerahkan penanganan percobaan ulang kepada “kebijaksanaan” komponen AI yang bertanggung jawab untuk memutuskan bagaimana menangani pengecualian tersebut.

3. **Strategi Fallback:** Jika sebuah komponen AI gagal memberikan respons yang valid atau mengalami error—kejadian yang umum mengingat sifatnya yang sangat baru—siapkan mekanisme fallback untuk memastikan alur kerja dapat terus berlanjut. Ini bisa melibatkan penggunaan nilai default, algoritma alternatif, atau **Human In The Loop** untuk membuat keputusan dan menjaga alur kerja tetap berjalan.
4. **Tindakan Kompensasi:** Arahan orkestrator harus mencakup instruksi tentang tindakan kompensasi untuk menangani eksepsi yang tidak dapat diselesaikan secara otomatis. Tindakan kompensasi adalah langkah-langkah yang diambil untuk membatalkan atau mengurangi dampak dari operasi yang gagal. Sebagai contoh, jika langkah pemrosesan pembayaran gagal, tindakan kompensasi bisa berupa pembatalan transaksi dan pemberitahuan kepada pengguna. Tindakan kompensasi membantu menjaga konsistensi dan integritas data ketika menghadapi eksepsi.
5. **Pemantauan dan Peringatan Eksepsi:** Siapkan mekanisme pemantauan dan peringatan untuk mendeteksi dan memberitahu pemangku kepentingan terkait tentang eksepsi-eksepsi kritis. Orkestrator dapat diatur untuk mengetahui ambang batas dan aturan untuk memicu peringatan ketika eksepsi melewati batas tertentu atau ketika jenis eksepsi tertentu terjadi. Hal ini memungkinkan identifikasi dan penyelesaian masalah secara proaktif sebelum berdampak pada keseluruhan sistem.

Berikut adalah contoh penanganan dan pemulihan eksepsi dalam komponen alur kerja

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
```

Dalam contoh ini, komponen `InventoryManager` memeriksa ketersediaan produk untuk pesanan tertentu. Jika jumlah yang tersedia tidak mencukupi, komponen akan memunculkan `InsufficientInventoryError`. Eksepsi tersebut ditangkap, dicatat, dan mekanisme percobaan ulang diimplementasikan. Jika batas percobaan ulang terlampaui, komponen akan beralih ke intervensi manual dengan memberi notifikasi kepada admin.

Dengan mengimplementasikan mekanisme penanganan dan pemulihan eksepsi yang kuat, Anda dapat memastikan bahwa alur kerja cerdas Anda tangguh, dapat dipelihara, dan mampu menangani situasi tak terduga dengan baik.

Pola-pola ini membentuk dasar orkestrasi alur kerja cerdas dan dapat dikombinasikan serta disesuaikan untuk memenuhi kebutuhan spesifik dari berbagai aplikasi. Dengan memanfaatkan pola-pola ini, pengembang dapat menciptakan alur kerja yang fleksibel, tangguh, dan dioptimalkan untuk kinerja dan pengalaman pengguna.

Pada bagian berikutnya, kita akan mengeksplorasi bagaimana pola-pola ini dapat diimplementasikan dalam praktik, menggunakan contoh dunia nyata dan cuplikan kode untuk mengilustrasikan integrasi komponen AI ke dalam manajemen alur kerja.

Mengimplementasikan Orkestrasi Alur Kerja Cerdas dalam Praktik

Setelah kita mengeksplorasi pola-pola kunci dalam orkestrasi alur kerja cerdas, mari kita dalami bagaimana pola-pola ini dapat diimplementasikan dalam aplikasi dunia nyata. Kita akan memberikan contoh praktis dan cuplikan kode untuk mengilustrasikan integrasi komponen AI ke dalam manajemen alur kerja.

Pemroses Pesanan Cerdas

Mari kita dalami contoh praktis implementasi orkestrasi alur kerja cerdas menggunakan komponen `OrderProcessor` berbasis AI dalam aplikasi e-commerce Ruby on Rails. `OrderProcessor` mewujudkan konsep [Process Manager Enterprise Integration](#) yang pertama kali kita temui di Bab 3 ketika membahas [Multitude of Workers](#). Komponen ini akan bertanggung jawab untuk mengelola alur kerja pemenuhan pesanan, membuat

keputusan perutean berdasarkan hasil antara, dan mengorkestrasi pelaksanaan berbagai tahap pemrosesan.

Proses pemenuhan pesanan melibatkan beberapa tahap seperti validasi pesanan, pemeriksaan inventaris, pemrosesan pembayaran, dan pengiriman. Setiap tahap diimplementasikan sebagai proses pekerja terpisah yang melakukan tugas spesifik dan mengembalikan hasil ke `OrderProcessor`. Tahap-tahap ini tidak wajib, dan bahkan tidak harus dilakukan dalam urutan yang tepat.

Berikut adalah contoh implementasi `OrderProcessor`. Ini menampilkan dua mixin dari `Raix`. Yang pertama (`ChatCompletion`) memberikannya kemampuan untuk melakukan penyelesaian chat, yang menjadikannya komponen AI. Yang kedua (`FunctionDispatch`) memungkinkan pemanggilan fungsi oleh AI, memungkinkannya merespons prompt dengan pemanggilan fungsi alih-alih pesan teks.

Fungsi-fungsi pekerja (`validate_order`, `check_inventory`, dan lainnya) mendelegasikan ke kelas-kelas pekerja masing-masing, yang bisa berupa komponen AI maupun non-AI, dengan satu-satunya persyaratan bahwa mereka mengembalikan hasil kerja mereka dalam format yang dapat direpresentasikan sebagai string.



Seperti semua contoh lain di bagian buku ini, kode ini praktis merupakan kode semu dan hanya dimaksudkan untuk menyampaikan makna dari pola tersebut dan menginspirasi kreasi Anda sendiri. Deskripsi lengkap tentang pola-pola dan contoh kode lengkap disertakan dalam Bagian 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_looping!` 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
```

Dalam contoh ini, `OrderProcessor` diinisialisasi dengan objek pesanan dan menyimpan transkrip dari eksekusi alur kerja, dalam format transkrip percakapan yang umum digunakan oleh model bahasa besar. AI diberikan kendali penuh untuk mengoordinasikan eksekusi berbagai tahap pemrosesan, seperti validasi pesanan, pemeriksaan inventaris, pemrosesan pembayaran, dan pengiriman.

Setiap kali metode `chat_completion` dipanggil, transkrip dikirim ke AI agar AI dapat memberikan hasil berupa pemanggilan fungsi. Sepenuhnya terserah AI untuk menganalisis hasil dari tahap sebelumnya dan menentukan tindakan yang tepat untuk diambil. Misalnya, jika pemeriksaan inventaris menunjukkan tingkat stok yang rendah, `OrderProcessor` dapat menjadwalkan tugas pengisian ulang. Jika pemrosesan pembayaran gagal, ia dapat memulai percobaan ulang atau memberi tahu layanan pelanggan.

Contoh di atas tidak memiliki fungsi yang didefinisikan untuk pengisian ulang atau pemberitahuan ke layanan pelanggan, tetapi tentu saja bisa ditambahkan.

Transkrip bertambah setiap kali fungsi dipanggil dan berfungsi sebagai catatan eksekusi alur kerja, termasuk hasil dari setiap tahap dan instruksi yang dihasilkan AI untuk langkah-langkah selanjutnya. Transkrip ini dapat digunakan untuk debugging, audit, dan memberikan visibilitas ke dalam proses pemenuhan pesanan.

Dengan memanfaatkan AI dalam `OrderProcessor`, aplikasi e-commerce dapat secara dinamis menyesuaikan alur kerja berdasarkan data real-time dan menangani pengecualian secara cerdas. Komponen AI dapat membuat keputusan yang tepat, mengoptimalkan alur kerja, dan memastikan pemrosesan pesanan berjalan lancar bahkan dalam skenario yang kompleks.

Fakta bahwa satu-satunya persyaratan pada proses pekerja adalah mengembalikan output yang dapat dipahami agar AI dapat mempertimbangkan apa yang harus dilakukan selanjutnya, mungkin mulai menyadarkan Anda bagaimana pendekatan ini dapat mengurangi pekerjaan pemetaan input/output yang biasanya terlibat ketika mengintegrasikan sistem-sistem yang berbeda satu sama lain.

Moderator Konten Cerdas

Aplikasi media sosial umumnya membutuhkan setidaknya moderasi konten minimal untuk memastikan komunitas yang aman dan sehat. Contoh komponen `Content-Moderator` ini memanfaatkan AI untuk mengoordinasikan alur kerja moderasi secara cerdas, membuat keputusan berdasarkan karakteristik konten dan hasil dari berbagai tahap moderasi.

Proses moderasi melibatkan beberapa tahap seperti analisis teks, pengenalan gambar, penilaian reputasi pengguna, dan peninjauan manual. Setiap tahap diimplementasikan

sebagai proses pekerja terpisah yang melakukan tugas spesifik dan mengembalikan hasilnya ke ContentModerator.

Berikut adalah contoh implementasi 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
```

Dalam contoh ini, ContentModerator diinisialisasi dengan objek konten dan mempertahankan transkrip moderasi dalam format percakapan. Komponen AI memiliki kendali penuh atas alur kerja moderasi, memutuskan langkah mana yang akan dijalankan berdasarkan karakteristik konten dan hasil dari setiap tahap.

Fungsi-fungsi pekerja yang tersedia untuk dipanggil oleh AI mencakup `analyze_text`, `recognize_image`, `assess_user_reputation`, dan `escalate_to_manual_review`. Setiap fungsi mendelegasikan tugas ke proses pekerja yang sesuai (TextAnalysisWorker, ImageRecognitionWorker, dll.) dan menambahkan hasilnya ke transkrip moderasi, dengan pengecualian fungsi eskalasi yang bertindak sebagai status akhir. Akhirnya, fungsi `approve_content` dan `reject_content` juga bertindak sebagai status akhir.

Komponen AI menganalisis konten dan menentukan tindakan yang tepat untuk diambil. Jika konten berisi referensi gambar, ia dapat memanggil pekerja `recognize_image` untuk bantuan dalam peninjauan visual. Jika ada pekerja yang memperingatkan tentang konten yang berpotensi berbahaya, AI dapat memutuskan untuk mengeskalasi konten untuk peninjauan manual atau langsung menolaknya. Tetapi tergantung pada tingkat keparahan peringatan, AI dapat memilih untuk menggunakan hasil penilaian

reputasi pengguna dalam memutuskan bagaimana menangani konten yang tidak yakin. Tergantung pada kasus penggunaan, mungkin pengguna yang terpercaya memiliki lebih banyak kelonggaran dalam apa yang dapat mereka posting. Dan seterusnya...

Seperti pada contoh pengelola proses sebelumnya, transkrip moderasi berfungsi sebagai catatan pelaksanaan alur kerja, termasuk hasil dari setiap langkah dan keputusan yang dihasilkan AI. Transkrip ini dapat digunakan untuk audit, transparansi, dan peningkatan proses moderasi dari waktu ke waktu.

Dengan memanfaatkan AI dalam ContentModerator, aplikasi media sosial dapat secara dinamis menyesuaikan alur kerja moderasi berdasarkan karakteristik konten dan menangani skenario moderasi yang kompleks secara cerdas. Komponen AI dapat membuat keputusan yang terinformasi, mengoptimalkan alur kerja, dan memastikan pengalaman komunitas yang aman dan sehat.

Mari kita jelajahi dua contoh lagi yang mendemonstrasikan penjadwalan tugas prediktif dan penanganan pengecualian serta pemulihan dalam konteks orkestrasi alur kerja cerdas.

Penjadwalan Tugas Prediktif dalam Sistem Dukungan Pelanggan

Dalam aplikasi dukungan pelanggan yang dibangun dengan Ruby on Rails, pengelolaan dan prioritas tiket dukungan yang efisien sangat penting untuk memberikan bantuan tepat waktu kepada pelanggan. Komponen `SupportTicketScheduler` memanfaatkan AI untuk memprediksi penjadwalan dan penugasan tiket dukungan kepada agen yang tersedia berdasarkan berbagai faktor seperti urgensi tiket, keahlian agen, dan beban kerja.

```
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: "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 end
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
```

Dalam contoh ini, `SupportTicketScheduler` diinisialisasi dengan objek tiket dukungan dan memelihara transkrip penjadwalan. Komponen AI menganalisis detail tiket dan secara prediktif menjadwalkan penugasan tiket berdasarkan faktor-faktor seperti urgensi tiket, keahlian agen, dan beban kerja agen yang diprediksi.

Fungsi-fungsi yang tersedia untuk digunakan AI mencakup `analyze_ticket_urgency`, `list_available_agents`, `predict_agent_workload`, dan `assign_ticket_to_agent`. Setiap fungsi mendelegasikan tugas ke komponen penganalisis atau prediktor yang sesuai dan menambahkan hasilnya ke transkrip penjadwalan. AI juga memiliki opsi untuk menunda penugasan menggunakan fungsi `delay_assignment`.

Komponen AI memeriksa transkrip penjadwalan dan membuat keputusan yang tepat mengenai penugasan tiket. Komponen ini mempertimbangkan urgensi tiket, keahlian agen yang tersedia, dan beban kerja yang diprediksi dari setiap agen untuk menentukan agen yang paling sesuai untuk menangani tiket tersebut.

Dengan memanfaatkan penjadwalan tugas prediktif, aplikasi dukungan pelanggan dapat mengoptimalkan penugasan tiket, mengurangi waktu respons, dan meningkatkan kepuasan pelanggan secara keseluruhan. Pengelolaan tiket dukungan yang proaktif dan efisien memastikan bahwa tiket yang tepat ditugaskan kepada agen yang tepat pada waktu yang tepat.

Penanganan Pengecualian dan Pemulihan dalam Saluran Pemrosesan Data

Penanganan pengecualian dan pemulihan dari kegagalan sangat penting untuk memastikan integritas data dan mencegah kehilangan data. Komponen `DataProcessingOrchestrator` menggunakan AI untuk menangani pengecualian secara cerdas dan mengoordinasikan proses pemulihan dalam saluran pemrosesan data

```
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        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
```

Dalam contoh ini, `DataProcessingOrchestrator` diinisialisasi dengan objek batch data dan mempertahankan transkrip pemrosesan. Komponen AI mengoordinasikan pipeline pemrosesan data, menangani eksepsi dan memulihkan dari kegagalan sesuai kebutuhan.

Fungsi-fungsi yang tersedia untuk digunakan AI mencakup `validate_data`, `process_data`, `request_fix`, `retry_processing`, dan `mark_data_as_failed`. Setiap fungsi mendelegasikan tugas ke komponen pemrosesan data yang sesuai dan menambahkan hasil atau detail eksepsi ke transkrip pemrosesan.

Jika terjadi eksepsi validasi selama tahap `validate_data`, fungsi `handle_validation_exception` akan menambahkan data eksepsi ke transkrip dan mengembalikan kendali ke AI. Demikian pula, jika terjadi eksepsi pemrosesan selama tahap `process_data`, AI dapat memutuskan strategi pemulihannya.

Tergantung pada sifat eksepsi yang ditemui, AI dapat memutuskan sesuai kebijakannya untuk memanggil `request_fix`, yang mendelegasikan ke komponen `Smart-`

DataFixer berbasis AI (lihat bab Self-Healing Data). Data fixer mendapatkan deskripsi dalam bahasa Inggris sederhana tentang bagaimana seharusnya memodifikasi `@data_batch` agar pemrosesan dapat dicoba ulang. Mungkin percobaan ulang yang berhasil akan melibatkan penghapusan catatan dari batch data yang gagal validasi dan/atau menyalinnya ke pipeline pemrosesan yang berbeda untuk ditinjau manusia? Kemungkinannya hampir tak terbatas.

Dengan memasukkan penanganan eksepsi dan pemulihan berbasis AI, aplikasi pemrosesan data menjadi lebih tangguh dan toleran terhadap kesalahan. `DataProcessingOrchestrator` secara cerdas mengelola eksepsi, meminimalkan kehilangan data, dan memastikan kelancaran eksekusi alur kerja pemrosesan data.

Pemantauan dan Pencatatan

Pemantauan dan pencatatan memberikan visibilitas terhadap kemajuan, kinerja, dan kesehatan komponen alur kerja berbasis AI, memungkinkan pengembang untuk melacak dan menganalisis perilaku sistem. Implementasi mekanisme pemantauan dan pencatatan yang efektif sangat penting untuk debugging, audit, dan perbaikan berkelanjutan dari alur kerja cerdas.

Memantau Kemajuan dan Kinerja Alur Kerja

Untuk memastikan kelancaran eksekusi alur kerja cerdas, penting untuk memantau kemajuan dan kinerja setiap komponen alur kerja. Ini melibatkan pelacakan metrik dan kejadian kunci sepanjang siklus hidup alur kerja.

Beberapa aspek penting yang perlu dipantau meliputi:

1. **Waktu Eksekusi Alur Kerja:** Mengukur waktu yang dibutuhkan oleh setiap komponen alur kerja untuk menyelesaikan tugasnya. Ini membantu mengidentifikasi bottleneck kinerja dan mengoptimalkan efisiensi alur kerja secara keseluruhan.

2. **Penggunaan Sumber Daya:** Memantau penggunaan sumber daya sistem, seperti CPU, memori, dan penyimpanan, oleh setiap komponen alur kerja. Ini membantu memastikan bahwa sistem beroperasi dalam kapasitasnya dan dapat menangani beban kerja secara efektif.

3. **Tingkat Kesalahan dan Eksepsi:** Lacak kejadian kesalahan dan eksepsi dalam komponen alur kerja. Hal ini membantu mengidentifikasi masalah potensial dan memungkinkan penanganan serta pemulihan kesalahan secara proaktif.

4. **Titik Keputusan dan Hasil:** Pantau titik-titik keputusan dalam alur kerja dan hasil dari keputusan berbasis AI. Ini memberikan wawasan tentang perilaku dan efektivitas komponen AI.

Data yang ditangkap oleh proses pemantauan dapat ditampilkan dalam dasbor atau digunakan sebagai masukan untuk laporan terjadwal yang menginformasikan administrator sistem tentang kesehatan sistem.



Data pemantauan dapat diumpankan ke proses administrator sistem berbasis AI untuk ditinjau dan ditindaklanjuti!

Pencatatan Log Kejadian dan Keputusan Penting

Pencatatan log adalah praktik penting yang melibatkan penangkapan dan penyimpanan informasi yang relevan tentang kejadian penting, keputusan, dan eksepsi yang terjadi selama eksekusi alur kerja.

Beberapa aspek penting yang perlu dicatat meliputi:

1. **Inisiasi dan Penyelesaian Alur Kerja:** Catat waktu mulai dan selesai setiap instansi alur kerja, beserta metadata yang relevan seperti data masukan dan konteks pengguna.

2. **Eksekusi Komponen:** Catat detail eksekusi setiap komponen alur kerja, termasuk parameter masukan, hasil keluaran, dan data perantara yang dihasilkan.

3. **Keputusan AI dan Penalaran:** Catat keputusan yang dibuat oleh komponen AI, beserta penalaran yang mendasarinya atau skor kepercayaan. Ini memberikan transparansi dan memungkinkan audit keputusan berbasis AI.

4. **Eksepsi dan Pesan Kesalahan:** Catat setiap eksepsi atau pesan kesalahan yang ditemui selama eksekusi alur kerja, termasuk jejak tumpukan dan informasi konteks yang relevan.

Pencatatan log dapat diimplementasikan menggunakan berbagai teknik, seperti menulis ke file log, menyimpan log dalam basis data, atau mengirim log ke layanan pencatatan log terpusat. Penting untuk memilih kerangka kerja pencatatan log yang menyediakan fleksibilitas, skalabilitas, dan integrasi yang mudah dengan arsitektur aplikasi.

Berikut adalah contoh bagaimana pencatatan log dapat diimplementasikan dalam aplikasi Ruby on Rails menggunakan kelas `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)
```

Dengan menempatkan pernyataan pencatatan secara strategis di seluruh komponen alur kerja dan titik keputusan AI, pengembang dapat mengumpulkan informasi berharga untuk debugging, audit, dan analisis.

Manfaat Pemantauan dan Pencatatan

Penerapan pemantauan dan pencatatan dalam orkestrasi alur kerja cerdas menawarkan beberapa manfaat:

1. **Debugging dan Pemecahan Masalah:** Log terperinci dan data pemantauan membantu pengembang mengidentifikasi dan mendiagnosis masalah dengan cepat. Keduanya memberikan wawasan tentang aliran eksekusi alur kerja, interaksi komponen, dan setiap kesalahan atau pengecualian yang ditemui.
2. **Optimasi Kinerja:** Pemantauan metrik kinerja memungkinkan pengembang mengidentifikasi bottleneck dan mengoptimalkan komponen alur kerja untuk efisiensi yang lebih baik. Dengan menganalisis waktu eksekusi, penggunaan sumber daya, dan metrik lainnya, pengembang dapat membuat keputusan yang tepat untuk meningkatkan kinerja keseluruhan sistem.
3. **Audit dan Kepatuhan:** Pencatatan kejadian dan keputusan penting menyediakan jejak audit untuk kepatuhan regulasi dan akuntabilitas. Hal ini memungkinkan organisasi untuk melacak dan memverifikasi tindakan yang diambil oleh komponen AI serta memastikan kepatuhan terhadap aturan bisnis dan persyaratan hukum.
4. **Perbaikan Berkelanjutan:** Data pemantauan dan pencatatan berfungsi sebagai masukan berharga untuk perbaikan berkelanjutan alur kerja cerdas. Dengan menganalisis data historis, mengidentifikasi pola, dan mengukur efektivitas keputusan AI, pengembang dapat menyempurnakan dan meningkatkan logika orkestrasi alur kerja secara iteratif.

Pertimbangan dan Praktik Terbaik

Saat menerapkan pemantauan dan pencatatan dalam orkestrasi alur kerja cerdas, pertimbangkan praktik terbaik berikut:

1. **Tentukan Metrik Pemantauan yang Jelas:** Identifikasi metrik dan kejadian utama

yang perlu dipantau berdasarkan persyaratan spesifik alur kerja. Fokus pada metrik yang memberikan wawasan bermakna tentang kinerja, kesehatan, dan perilaku sistem.

2. Terapkan Pencatatan Terperinci: Pastikan pernyataan pencatatan ditempatkan pada titik-titik yang tepat dalam komponen alur kerja dan titik keputusan AI. Tangkap informasi konteks yang relevan, seperti parameter input, hasil output, dan data perantara yang dihasilkan.

3. Gunakan Pencatatan Terstruktur: Adopsi format pencatatan terstruktur untuk memfasilitasi penguraian dan analisis data log dengan mudah. Pencatatan terstruktur memungkinkan pencarian, penyaringan, dan pengumpulan entri log yang lebih baik.

4. Kelola Retensi dan Rotasi Log: Terapkan kebijakan retensi dan rotasi log untuk mengelola penyimpanan dan siklus hidup file log. Tentukan periode retensi yang sesuai berdasarkan persyaratan hukum, batasan penyimpanan, dan kebutuhan analisis. Jika memungkinkan, alihkan pencatatan ke layanan pihak ketiga seperti [Papertrail](#).

5. Mengamankan Informasi Sensitif: Berhati-hatilah saat mencatat informasi sensitif, seperti informasi identitas pribadi (PII) atau data bisnis yang bersifat rahasia. Terapkan langkah-langkah keamanan yang sesuai, seperti penyamaran data atau enkripsi, untuk melindungi informasi sensitif dalam berkas log.

6. Integrasikan dengan Perangkat Pemantauan dan Peringatan: Manfaatkan perangkat pemantauan dan peringatan untuk memusatkan pengumpulan, analisis, dan visualisasi data pemantauan dan pencatatan. Perangkat-perangkat ini dapat memberikan wawasan real-time, menghasilkan peringatan berdasarkan ambang batas yang telah ditentukan, dan memfasilitasi deteksi dan penyelesaian masalah secara proaktif. Perangkat favorit saya untuk hal ini adalah [Datadog](#).

Dengan menerapkan mekanisme pemantauan dan pencatatan yang komprehensif, pengembang dapat memperoleh wawasan berharga tentang perilaku dan kinerja alur kerja cerdas. Wawasan ini memungkinkan debugging yang efektif, optimalisasi, dan perbaikan berkelanjutan sistem orkestrasi alur kerja berbasis AI.

Pertimbangan Skalabilitas dan Kinerja

Skalabilitas dan kinerja adalah aspek-aspek penting yang perlu dipertimbangkan saat merancang dan mengimplementasikan sistem orkestrasi alur kerja cerdas. Seiring meningkatnya volume alur kerja yang berjalan bersamaan dan kompleksitas komponen berbasis AI, menjadi penting untuk memastikan bahwa sistem dapat menangani beban kerja secara efisien dan dapat meningkat secara mulus untuk memenuhi permintaan yang terus bertambah.

Menangani Volume Tinggi Alur Kerja yang Berjalan Bersamaan

Sistem orkestrasi alur kerja cerdas sering kali perlu menangani sejumlah besar alur kerja yang berjalan bersamaan. Untuk memastikan skalabilitas, pertimbangkan strategi-strategi berikut:

1. **Pemrosesan Asinkron:** Terapkan mekanisme pemrosesan asinkron untuk memisahkan eksekusi komponen alur kerja. Ini memungkinkan sistem untuk menangani beberapa alur kerja secara bersamaan tanpa memblokir atau menunggu setiap komponen selesai. Pemrosesan asinkron dapat dicapai menggunakan antrian pesan, arsitektur berbasis event, atau kerangka kerja pemrosesan tugas latar belakang seperti Sidekiq.
2. **Arsitektur Terdistribusi:** Rancang arsitektur sistem untuk menggunakan komponen serverless (seperti AWS Lambda) atau cukup distribusikan beban kerja di beberapa node atau server bersamaan dengan server aplikasi utama Anda. Ini memungkinkan skalabilitas horizontal, di mana node tambahan dapat ditambahkan untuk menangani peningkatan volume alur kerja.
3. **Eksekusi Paralel:** Identifikasi peluang untuk eksekusi paralel dalam alur kerja. Beberapa komponen alur kerja mungkin tidak bergantung satu sama lain dan dapat

dieksekusi secara bersamaan. Dengan memanfaatkan teknik pemrosesan paralel, seperti multi-threading atau antrian tugas terdistribusi, sistem dapat mengoptimalkan penggunaan sumber daya dan mengurangi waktu eksekusi alur kerja secara keseluruhan.

Mengoptimalkan Kinerja Komponen Berbasis AI

Komponen berbasis AI, seperti model pembelajaran mesin atau mesin pemrosesan bahasa alami, dapat membutuhkan komputasi intensif dan mempengaruhi kinerja keseluruhan sistem orkestrasi alur kerja. Untuk mengoptimalkan kinerja komponen AI, pertimbangkan teknik-teknik berikut:

- 1. Penyimpanan Cache:** Jika pemrosesan AI Anda bersifat murni generatif dan tidak melibatkan pencarian informasi realtime atau integrasi eksternal untuk menghasilkan chat completion, maka Anda dapat mempertimbangkan mekanisme penyimpanan cache untuk menyimpan dan menggunakan kembali hasil dari operasi yang sering diakses atau membutuhkan komputasi berat.
- 2. Optimasi Model:** Terus optimalkan cara Anda menggunakan model AI dalam komponen alur kerja. Ini mungkin melibatkan teknik seperti *Prompt Distillation* atau mungkin hanya sebatas menguji model-model baru saat tersedia.
- 3. Pemrosesan Batch:** Jika Anda bekerja dengan model kelas GPT-4, Anda mungkin dapat memanfaatkan teknik pemrosesan batch untuk memproses beberapa data atau permintaan dalam satu batch, alih-alih memprosesnya satu per satu. Dengan memproses data dalam batch, sistem dapat mengoptimalkan penggunaan sumber daya dan mengurangi beban dari permintaan model yang berulang.

Pemantauan dan Profiling Kinerja

Untuk mengidentifikasi bottleneck kinerja dan mengoptimalkan skalabilitas sistem orkestrasi alur kerja cerdas, sangat penting untuk menerapkan mekanisme pemantauan dan profiling. Pertimbangkan pendekatan berikut:

- 1. Metrik Kinerja:** Tentukan dan pantau metrik kinerja utama, seperti waktu respons, throughput, penggunaan sumber daya, dan latensi. Metrik ini memberikan wawasan tentang kinerja sistem dan membantu mengidentifikasi area yang perlu dioptimalkan. Agregator model AI populer [OpenRouter](#) menyertakan metrik Host¹ dan Speed² dalam setiap respons API, memudahkan pelacakan metrik-metrik utama ini.
- 2. Alat Profiling:** Gunakan alat profiling untuk menganalisis kinerja komponen alur kerja individu dan operasi AI. Alat profiling dapat membantu mengidentifikasi titik-titik kritis kinerja, jalur kode yang tidak efisien, atau operasi yang membutuhkan banyak sumber daya. Alat profiling populer termasuk New Relic, Scout, atau profiler bawaan yang disediakan oleh bahasa pemrograman atau framework.
- 3. Pengujian Beban:** Lakukan pengujian beban untuk mengevaluasi kinerja sistem di bawah berbagai tingkat beban kerja konkuren. Pengujian beban membantu mengidentifikasi batas skalabilitas sistem, mendeteksi penurunan kinerja, dan memastikan bahwa sistem dapat menangani lalu lintas yang diharapkan tanpa mengorbankan kinerja.
- 4. Pemantauan Berkelanjutan:** Terapkan mekanisme pemantauan dan peringatan berkelanjutan untuk mendeteksi masalah kinerja dan bottleneck secara proaktif. Siapkan dashboard pemantauan dan peringatan untuk melacak indikator kinerja utama (KPI) dan menerima notifikasi ketika ambang batas yang telah ditentukan terlampaui. Hal ini memungkinkan identifikasi dan penyelesaian masalah kinerja secara cepat.

Strategi Penskalaan

Untuk menangani beban kerja yang meningkat dan memastikan skalabilitas sistem orkestrasi alur kerja cerdas, pertimbangkan strategi penskalaan berikut:

- 1. Penskalaan Vertikal:** Penskalaan vertikal melibatkan peningkatan sumber daya (misalnya, CPU, memori) dari node atau server individual untuk menangani beban kerja

¹Host adalah waktu yang diperlukan untuk menerima byte pertama dari generasi yang distream dari host model, atau dikenal sebagai “time to first byte.”

²Speed dihitung sebagai jumlah token completion dibagi dengan total waktu generasi. Untuk permintaan non-stream, latensi dianggap sebagai bagian dari waktu generasi.

yang lebih tinggi. Pendekatan ini cocok ketika sistem membutuhkan daya pemrosesan atau memori yang lebih besar untuk menangani alur kerja kompleks atau operasi AI.

2. Penskalaan Horizontal: Penskalaan horizontal melibatkan penambahan lebih banyak node atau server ke sistem untuk mendistribusikan beban kerja. Pendekatan ini efektif ketika sistem perlu menangani sejumlah besar alur kerja konkuren atau ketika beban kerja dapat dengan mudah didistribusikan ke beberapa node. Penskalaan horizontal membutuhkan arsitektur terdistribusi dan mekanisme penyeimbangan beban untuk memastikan distribusi lalu lintas yang merata.

3. Auto-Scaling: Terapkan mekanisme auto-scaling untuk menyesuaikan jumlah node atau sumber daya secara otomatis berdasarkan permintaan beban kerja. Auto-scaling memungkinkan sistem untuk melakukan penskalaan naik atau turun secara dinamis tergantung pada lalu lintas yang masuk, memastikan penggunaan sumber daya yang optimal dan efisiensi biaya. Platform cloud seperti Amazon Web Services (AWS) atau Google Cloud Platform (GCP) menyediakan kemampuan auto-scaling yang dapat dimanfaatkan untuk sistem orkestrasi alur kerja cerdas.

Teknik Optimasi Kinerja

Selain strategi penskalaan, pertimbangkan teknik optimasi kinerja berikut untuk meningkatkan efisiensi sistem orkestrasi alur kerja cerdas:

1. Penyimpanan dan Pengambilan Data yang Efisien: Optimalkan mekanisme penyimpanan dan pengambilan data yang digunakan oleh komponen alur kerja. Gunakan pengindeksan database yang efisien, teknik optimasi query, dan caching data untuk meminimalkan latensi dan meningkatkan kinerja operasi yang intensif data.

2. I/O Asinkron: Manfaatkan operasi I/O asinkron untuk mencegah pemblokiran dan meningkatkan daya tanggap sistem. I/O asinkron memungkinkan sistem untuk menangani beberapa permintaan secara bersamaan tanpa menunggu operasi I/O selesai, sehingga memaksimalkan penggunaan sumber daya.

3. **Serialisasi dan Deserialisasi yang Efisien:** Optimalkan proses serialisasi dan deserialisasi yang digunakan untuk pertukaran data antara komponen alur kerja. Gunakan format serialisasi yang efisien, seperti Protocol Buffers atau MessagePack, untuk mengurangi beban serialisasi data dan meningkatkan kinerja komunikasi antar-komponen.



Untuk aplikasi berbasis Ruby, pertimbangkan untuk menggunakan [Universal ID](#). Universal ID memanfaatkan MessagePack dan Brotli (kombinasi yang dirancang untuk kecepatan dan kompresi data terbaik di kelasnya). Ketika dikombinasikan, pustaka-pustaka ini 30% lebih cepat dan memiliki tingkat kompresi dalam rentang 2-5% dibandingkan dengan Protocol Buffers.

4. **Kompresi dan Pengkodean:** Terapkan teknik kompresi dan pengkodean untuk mengurangi ukuran data yang ditransfer antara komponen alur kerja. Algoritma kompresi, seperti gzip atau Brotli, dapat secara signifikan mengurangi penggunaan bandwidth jaringan dan meningkatkan kinerja sistem secara keseluruhan.

Dengan mempertimbangkan aspek skalabilitas dan kinerja selama perancangan dan implementasi sistem orkestrasi alur kerja cerdas, Anda dapat memastikan bahwa sistem Anda dapat menangani volume alur kerja konkuren yang tinggi, mengoptimalkan kinerja komponen berbasis AI, dan dapat diskalakan dengan mulus untuk memenuhi permintaan yang terus berkembang. Pemantauan berkelanjutan, profiling, dan upaya optimasi sangat penting untuk mempertahankan kinerja dan responsivitas sistem seiring dengan peningkatan beban kerja dan kompleksitas dari waktu ke waktu.

Pengujian dan Validasi Alur Kerja

Pengujian dan validasi merupakan aspek penting dalam pengembangan dan pemeliharaan sistem orkestrasi alur kerja cerdas. Mengingat sifat kompleks dari alur kerja berbasis AI, sangat penting untuk memastikan bahwa setiap komponen berfungsi sesuai

harapan, alur kerja secara keseluruhan berperilaku dengan benar, dan keputusan AI akurat dan dapat diandalkan. Dalam bagian ini, kita akan mengeksplorasi berbagai teknik dan pertimbangan untuk menguji dan memvalidasi alur kerja cerdas.

Pengujian Unit Komponen Alur Kerja

Pengujian unit melibatkan pengujian komponen alur kerja secara individual untuk memverifikasi kebenaran dan ketahanannya. Saat melakukan pengujian unit pada komponen alur kerja berbasis AI, pertimbangkan hal-hal berikut:

1. **Validasi Masukan:** Uji kemampuan komponen dalam menangani berbagai jenis masukan, termasuk data yang valid dan tidak valid. Verifikasi bahwa komponen dapat menangani kasus-kasus ekstrem dengan baik dan memberikan pesan kesalahan atau pengecualian yang sesuai.
2. **Verifikasi Keluaran:** Pastikan bahwa komponen menghasilkan keluaran yang diharapkan untuk sekumpulan masukan tertentu. Bandingkan keluaran aktual dengan hasil yang diharapkan untuk memastikan kebenarannya.
3. **Penanganan Kesalahan:** Uji mekanisme penanganan kesalahan komponen dengan mensimulasikan berbagai skenario kesalahan, seperti input tidak valid, ketidakterdediaan sumber daya, atau eksepsi tak terduga. Verifikasi bahwa komponen menangkap dan menangani kesalahan dengan tepat.
4. **Kondisi Batas:** Uji perilaku komponen dalam kondisi batas, seperti input kosong, ukuran input maksimum, atau nilai-nilai ekstrem. Pastikan bahwa komponen menangani kondisi-kondisi ini dengan baik tanpa mengalami crash atau menghasilkan hasil yang tidak benar.

Berikut adalah contoh uji unit untuk komponen alur kerja dalam Ruby menggunakan kerangka kerja pengujian 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
```

Dalam contoh ini, komponen `OrderValidator` diuji menggunakan dua kasus uji: satu untuk pesanan yang valid dan satu lagi untuk pesanan yang tidak valid. Kasus-kasus uji ini memverifikasi bahwa metode `validate` mengembalikan nilai boolean yang diharapkan berdasarkan validitas pesanan.

Interaksi Alur Kerja Pengujian Integrasi

Pengujian integrasi berfokus pada verifikasi interaksi dan aliran data antara berbagai komponen alur kerja. Ini memastikan bahwa komponen-komponen bekerja bersama dengan lancar dan menghasilkan luaran yang diharapkan. Ketika melakukan pengujian integrasi pada alur kerja cerdas, pertimbangkan hal-hal berikut:

1. **Interaksi Komponen:** Uji komunikasi dan pertukaran data antar komponen alur kerja. Verifikasi bahwa keluaran dari satu komponen dioper dengan benar sebagai masukan ke komponen berikutnya dalam alur kerja.

2. **Konsistensi Data:** Pastikan data tetap konsisten dan akurat saat mengalir melalui alur kerja. Verifikasi bahwa transformasi data, perhitungan, dan agregasi dilakukan dengan benar.

3. **Propagasi Pengecualian:** Uji bagaimana pengecualian dan kesalahan dipropagasikan dan ditangani di seluruh komponen alur kerja. Verifikasi bahwa pengecualian ditangkap, dicatat, dan ditangani dengan tepat untuk mencegah gangguan alur kerja.

4. **Perilaku Asinkron:** Jika alur kerja melibatkan komponen asinkron atau eksekusi paralel, uji mekanisme koordinasi dan sinkronisasinya. Pastikan bahwa alur kerja berperilaku dengan benar dalam skenario konkuren dan asinkron.

Berikut adalah contoh pengujian integrasi untuk alur kerja dalam Ruby menggunakan kerangka kerja pengujian 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
```

Dalam contoh ini, `OrderProcessingWorkflow` diuji dengan memverifikasi interaksi antara berbagai komponen alur kerja. Kasus pengujian ini menyiapkan ekspektasi untuk perilaku setiap komponen dan memastikan bahwa alur kerja memproses pesanan dengan sukses, serta memperbarui status pesanan dengan sesuai.

Pengujian Titik Keputusan AI

Pengujian titik keputusan AI sangat penting untuk memastikan akurasi dan keandalan alur kerja berbasis AI. Saat menguji titik keputusan AI, pertimbangkan hal-hal berikut:

1. **Akurasi Keputusan:** Verifikasi bahwa komponen AI membuat keputusan yang akurat berdasarkan data masukan dan model yang telah dilatih. Bandingkan keputusan AI dengan hasil yang diharapkan atau data acuan kebenaran.
2. **Kasus-kasus Ekstrim:** Uji perilaku komponen AI dalam kasus-kasus ekstrim dan skenario tidak biasa. Verifikasi bahwa komponen AI menangani kasus-kasus ini dengan baik dan membuat keputusan yang masuk akal.
3. **Bias dan Keadilan:** Evaluasi komponen AI untuk potensi bias dan pastikan bahwa komponen tersebut membuat keputusan yang adil dan tidak bias. Uji komponen dengan data masukan yang beragam dan analisis hasilnya untuk mencari pola-pola diskriminatif.
4. **Kemampuan Menjelaskan:** Jika komponen AI memberikan penjelasan atau alasan untuk keputusannya, verifikasi kebenaran dan kejelasan penjelasan tersebut. Pastikan bahwa penjelasan tersebut selaras dengan proses pengambilan keputusan yang mendasarinya.

Berikut adalah contoh pengujian titik keputusan AI dalam Ruby menggunakan kerangka kerja pengujian 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
```

Dalam contoh ini, komponen AI FraudDetector diuji dengan dua kasus pengujian: satu untuk transaksi penipuan dan satu lagi untuk transaksi yang sah. Kasus-kasus pengujian ini memverifikasi bahwa metode `detect_fraud` mengembalikan nilai boolean yang diharapkan berdasarkan karakteristik transaksi.

Pengujian End-to-End

Pengujian end-to-end melibatkan pengujian seluruh alur kerja dari awal hingga akhir, mensimulasikan skenario dunia nyata dan interaksi pengguna. Ini memastikan bahwa alur kerja berperilaku dengan benar dan menghasilkan hasil yang diinginkan. Ketika melakukan pengujian end-to-end untuk alur kerja cerdas, pertimbangkan hal-hal berikut:

1. **Skenario Pengguna:** Identifikasi skenario pengguna yang umum dan uji perilaku alur kerja dalam skenario-skenario tersebut. Verifikasi bahwa alur kerja menangani input pengguna dengan benar, membuat keputusan yang tepat, dan menghasilkan output yang diharapkan.
2. **Validasi Data:** Pastikan bahwa alur kerja memvalidasi dan membersihkan input pengguna untuk mencegah inkonsistensi data atau kerentanan keamanan. Uji alur kerja dengan berbagai jenis data input, termasuk data yang valid dan tidak valid.
3. **Pemulihan Kesalahan:** Uji kemampuan alur kerja untuk pulih dari kesalahan dan pengecualian. Simulasikan skenario kesalahan dan verifikasi bahwa alur kerja menanganinya dengan baik, mencatat kesalahan, dan mengambil tindakan pemulihan yang tepat.
4. **Kinerja dan Skalabilitas:** Nilai kinerja dan skalabilitas alur kerja dalam berbagai kondisi beban. Uji alur kerja dengan volume permintaan bersamaan yang besar dan ukur waktu respons, penggunaan sumber daya, dan stabilitas sistem secara keseluruhan.

Berikut adalah contoh pengujian end-to-end untuk alur kerja dalam Ruby menggunakan kerangka kerja pengujian RSpec dan pustaka Capybara untuk mensimulasikan interaksi pengguna:

```
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
13 end
```

Dalam contoh ini, pengujian end-to-end mensimulasikan pengguna yang melakukan

pemesanan melalui antarmuka web. Pengujian ini mengisi bidang-bidang formulir yang diperlukan, mengirimkan pesanan, dan memverifikasi bahwa pesanan diproses dengan sukses, menampilkan pesan konfirmasi yang sesuai dan memperbarui status pesanan dalam database.

Integrasi dan Penerapan Berkelanjutan

Untuk memastikan keandalan dan kemudahan pemeliharaan alur kerja cerdas, disarankan untuk mengintegrasikan pengujian dan validasi ke dalam pipeline Integrasi dan Penerapan Berkelanjutan (CI/CD). Hal ini memungkinkan pengujian dan validasi otomatis terhadap perubahan alur kerja sebelum diterapkan ke produksi. Pertimbangkan praktik-praktik berikut:

1. **Eksekusi Pengujian Otomatis:** Mengonfigurasi pipeline CI/CD untuk menjalankan rangkaian pengujian secara otomatis setiap kali ada perubahan pada basis kode alur kerja. Ini memastikan bahwa setiap regresi atau kegagalan dapat terdeteksi sejak dini dalam proses pengembangan.
2. **Pemantauan Cakupan Pengujian:** Mengukur dan memantau cakupan pengujian dari komponen alur kerja dan titik keputusan AI. Targetkan cakupan pengujian yang tinggi untuk memastikan bahwa jalur dan skenario kritis telah diuji secara menyeluruh.
3. **Umpan Balik Berkelanjutan:** Mengintegrasikan hasil pengujian dan metrik kualitas kode ke dalam alur kerja pengembangan. Berikan umpan balik berkelanjutan kepada pengembang tentang status pengujian, kualitas kode, dan masalah apa pun yang terdeteksi selama proses CI/CD.
4. **Lingkungan Staging:** Menerapkan alur kerja ke lingkungan staging yang mencerminkan lingkungan produksi secara akurat. Lakukan pengujian dan validasi tambahan di lingkungan staging untuk menangkap masalah yang terkait dengan infrastruktur, konfigurasi, atau integrasi data.
5. **Mekanisme Rollback:** Menerapkan mekanisme rollback untuk mengantisipasi kegagalan penerapan atau masalah kritis yang terdeteksi di produksi. Pastikan bahwa alur

kerja dapat dengan cepat dikembalikan ke versi stabil sebelumnya untuk meminimalkan waktu henti dan dampak terhadap pengguna.

Dengan mengintegrasikan pengujian dan validasi di sepanjang siklus pengembangan alur kerja cerdas, organisasi dapat memastikan keandalan, akurasi, dan kemudahan pemeliharaan sistem berbasis AI mereka. Pengujian dan validasi rutin membantu menangkap bug, mencegah regresi, dan membangun kepercayaan terhadap perilaku dan hasil alur kerja.

Bagian 2: Pola-Pola

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Rekayasa Prompt

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Rantai Pemikiran

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh-contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pembuatan Konten

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pembuatan Entitas Terstruktur

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Panduan Agen LLM

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Manfaat dan Pertimbangan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pengalihan Mode

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kapan Menggunakannya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Penugasan Peran

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kapan Menggunakannya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Objek Prompt

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Prompt Template

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Manfaat dan Pertimbangan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kapan Menggunakannya:

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

IO Terstruktur

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Menskalakan Structured IO

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Manfaat dan Pertimbangan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Perangkaian Prompt

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kapan Menggunakannya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh: Orientasi Olympia

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Penulis Ulang Prompt

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Response Fencing

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Manfaat dan Pertimbangan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Penanganan Kesalahan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Query Analyzer

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Implementasi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Penandaan Kelas Kata (POS) dan Pengenalan Entitas Bernama (NER)

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Klasifikasi Maksud

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Ekstraksi Kata Kunci

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Manfaat

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pengolah Ulang Kueri

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Manfaat

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Ventriloquist

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kapan Menggunakannya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Komponen Diskret

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Predicate

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kapan Menggunakannya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

API Facade

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Manfaat Utama

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kapan Menggunakannya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Autentikasi dan Otorisasi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Penanganan Permintaan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pemformatan Respons

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Penanganan Kesalahan dan Kasus-kasus Khusus

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pertimbangan Skalabilitas dan Kinerja

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Perbandingan dengan Pola Desain Lainnya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Penerjemah Hasil

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kapan Menggunakannya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Mesin Virtual

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kapan Menggunakannya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Di Balik Keajaiban

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Spesifikasi dan Pengujian

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Menentukan Perilaku

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Menulis Kasus Uji

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh: Pengujian Komponen Penerjemah

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pengulangan Interaksi HTTP

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Human In The Loop (HITL)

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pola-Pola Tingkat Tinggi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kecerdasan Hibrida

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Respons Adaptif

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Peralihan Peran Manusia-AI

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Eskalasi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Manfaat Utama

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Penerapan Dunia Nyata: Layanan Kesehatan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Siklus Umpan Balik

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Aplikasi dan Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Teknik Lanjutan dalam Integrasi Umpan Balik Manusia

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Radiasi Informasi Pasif

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Tampilan Informasi Kontekstual

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Notifikasi Proaktif

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Wawasan Penjelasan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Eksplorasi Interaktif

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Manfaat Utama

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Aplikasi dan Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Collaborative Decision Making (CDM)

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pembelajaran Berkelanjutan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Aplikasi dan Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pertimbangan Etis

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Peran HITL dalam Mitigasi Risiko AI

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kemajuan Teknologi dan Prospek Masa Depan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Tantangan dan Keterbatasan Sistem HITL

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Penanganan Kesalahan Cerdas

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pendekatan Tradisional dalam Penanganan Kesalahan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Diagnosis Kesalahan Kontekstual

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Rekayasa Prompt untuk Diagnosis Kesalahan Kontekstual

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Retrieval-Augmented Generation untuk Diagnosis Kesalahan Kontekstual

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pelaporan Kesalahan Cerdas

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pencegahan Error Prediktif

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pemulihan Kesalahan Cerdas

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Komunikasi Kesalahan Terpersonalisasi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Alur Kerja Penanganan Kesalahan Adaptif

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Kendali Mutu

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Eval

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Masalah

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Solusi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pertimbangan-pertimbangan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Memahami Referensi Emas

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerja Evaluasi Bebas Referensi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pagar Pengaman

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Masalah

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Solusi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Cara Kerjanya

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Contoh

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Pertimbangan

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Guardrails dan Evaluasi: Dua Sisi Mata Uang yang Sama

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Ketertukaraan Pagar Pengaman dan Evaluasi Bebas Referensi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Mengimplementasikan Pagar Pengaman dan Evaluasi Dwifungsi

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Glosarium

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Glosarium

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

A

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

B

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

C

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

D

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

E

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

F

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

G

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

H

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

I

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

J

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

K

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

L

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

M

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

N

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

O

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

P

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Q

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

R

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

S

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

T

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

U

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

V

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

W

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Z

Konten ini tidak tersedia dalam buku sampel. Buku tersebut dapat dibeli di Leanpub pada <http://leanpub.com/patterns-of-application-development-using-ai-id>.

Index

- Agentik, 32
- AI, 66, 76, 101, 130, 136, 144, 152, 204, 212
 - aplikasi, 127, 140, 151, 164
 - konversasional, 7, 214
 - model, 91, 101, 157, 158, 160, 212
 - percakapan, 31
 - sistem majemuk, 30, 31, 34
 - titik keputusan, 259
- aksesibilitas, 219, 220
- akun, 93
- aljabar linier, 43
- Alpaca, 13
- Altman, Sam, 18
- alur kerja adaptif
 - Komposisi Alur Kerja Adaptif, 228
- alur kerja bertahap, 113
- Amazon Web Services, 254
- analisis sentimen, 17, 102, 114–116, 119,
 - 120, 136, 146
- Analisis Sentimen Pelanggan, 102
- antarmuka inklusif, 201
- antarmuka kendali suara, 34
- Antarmuka Pengguna (UI)
 - antarmuka, 200, 216
 - kerangka kerja, 216
 - teknologi, 211
- Antarmuka Pengguna Generatif (GenUI),
 - 200
 - antarmuka visual, 211
- Anthropic, 23, 39, 75, 131, 138
- antropomorfisme, 70
- API, 73, 125, 155
- aplikasi chatbot, 121
- Aplikasi E-commerce, 93
- aplikasi modern, 225
- aplikasi pendidikan, 32
- arrays, 132
- arsitektur aplikasi enterprise, 38
- arsitektur berbasis event, 110
- Arsitektur Mikroservis, 91
- arsitektur perangkat lunak, 2
- arsitektur terdistribusi, 251
- arsitektur transformer, 6
- asisten virtual, 34
- aturan bisnis, 224
- aturan tata bahasa, 4
- audit dan kepatuhan, 249
- augmented reality glasses, 221
- auto-scaling, 254
- bahasa
 - Deteksi Bahasa, 114
 - model, 43, 75

- bahasa alami
 - Pemrosesan Bahasa Alami (NLP), 103
- bahasa natural
 - Pemrosesan Bahasa Natural (NLP), 122
- baris perintah
 - Antarmuka Baris Perintah (CLI), 26
- basis data, 125
 - objek berbasis, 107
- basis pengetahuan, 7
- basis pengetahuan Olympia, 93
- BERT, 14, 24
- bias
 - dan keadilan dalam AI, 259
- Brotli, 255
- Byte Pair Encoding (BPE), 15

- C (Bahasa Pemrograman), 118
- Chain of Thought (CoT), 45
- chatbot layanan pelanggan, 34
- ChatGPT, 30, 54
- Claude, 8, 44, 79
- Claude 3, 50, 128, 131, 136, 138
- Claude 3 Opus, 76
- Claude v1, 17
- Claude v2, 17
- Cohere (LLM Provider), 25
- Cohere (Penyedia LLM), 23
- concurrent workflows, 255
- context
 - Augmentation, 46
 - Contextual Content Generation, 193–195
- conversation
 - transcript, 159

- data
 - aliran, 112
 - analisis, 34, 148
 - integritas, 242
 - Pengambilan Data, 111
 - persiapan, 111
 - persistensi, 111
 - privasi, 27, 218
 - saluran pemrosesan, 242
 - Sinkronisasi Data, 111
 - tugas pemrosesan, 127
 - Validasi Data, 261
- data pelatihan, 42
- Data Self-Healing, 246
- data streaming, 154
- data terstruktur, 136
- Data yang Dapat Memperbaiki Diri, 166
- database
 - strategi penguncian, 112
- Datadog, 250
- debugging
 - and testing, 134
 - dan pemecahan masalah, 249
- desain aplikasi dan kerangka kerja, 201
- desktop computers, 221
- deteksi penipuan
 - sistem, 99
- dictionaries, 132

- digital landscape, 196
- direktif sistem, 130
- Dohan, et al., 44
- Dukungan Keputusan Klinis, 105
- dukungan pelanggan, 32
- Dynamic Tool Selection, 133

- e-commerce, 194, 224
- efisiensi, 225
- ekosistem, 149
- eksekusi paralel, 252
- eksperimen
 - kerangka, 196
- ELK stack, 112
- ensembel, 120
 - ensembel pekerja, 120
- ensemble, 119
- Enterprise Integration Patterns, 106
- error
 - penanganan, 144
- errors
 - Intelligent Error Handling, 144
- etika
 - implikasi, 201

- F#, 95
- Facebook, 24
- faktor risiko, 97, 98
- few-shot
 - pembelajaran, 63
 - prompting, 64
- fine-tuning, 82
- FitAI, 213

- fleksibilitas dan kreativitas, 198
- Forced Tool Selection, 133
- fungsi
 - nama, 156
 - pemanggilan, 125, 159
 - riwayat pemanggilan, 158

- Gemma 7B, 11
- Generasi Berbasis Pengambilan (RAG), 32
- generasi lintas modal, 22
- Generasi yang Diperkaya Pengambilan (RAG), 127
- Generative Pre-trained Transformer (GPT),
 - 8, 69
- Generative UI (GenUI), 207, 208
- GitLab, 95
- Global Interpreter Lock (GIL), 117
- Google, 23
 - API, 64, 66
 - Cloud AI Platform, 24
 - Cloud Platform, 254
 - Gemini, 21
 - Gemini 1.5 Pro, 14, 17, 18
 - PaLM (Pathways Language Model),
 - 17, 24
 - T5, 14
- GPT-3, 13, 17
- GPT-4, 6, 13, 17, 21, 31, 44, 50, 64, 107, 119,
 - 122, 129, 135, 206, 252
- Graham, Paul, 19
- GraphQL, 110
- Groq, 26, 122

- gzip, 255
- hambatan, 228
- hash, 154
- Hohpe, Gregor, 106
- Honeybadger, 96
- HTTP, 152
- Human-In-The-Loop (HITL), 181
- hyperparameter, 47
- identifikasi topik, 122
- ilmu komputer, 72, 74
- Inferensi, 5
- informasi
 - ekstraksi, 54
 - pengambilan, 7, 128
- injeksi SQL, 72
- input
 - prompts, 57
 - validation, 256
- Integrasi dan Penerapan Berkelanjutan (CI/CD), 262
 - pipeline, 262
- intelligent workflow orchestration, 255
- interaksi gaya bermain peran, 7
- internasionalisasi, 197
- Interpreter Hasil, 143
- intervensi manual, 230
- iterative refinement, 145
- jaringan saraf, 4, 6
- JSON (JavaScript Object Notation), 128, 132, 133, 136, 149, 169
- K-means, 124
- karyawan Databricks, 53
- kasus-kasus khusus, 59
- kegagalan pemanggilan fungsi, 135
- Kelanjutan Otomatis, 162
- kemampuan menjelaskan, 259
- kemampuan pengambilan keputusan, 101
- kepercayaan pengguna, 219
- keputusan
 - pohon, 224
 - titik, 247
- kerangka kerja pengembangan, 150
- kesalahan
 - pemulihan, 261
 - penanganan, 109, 112, 256
 - tingkat, 112
- kinerja
 - masalah, 253
 - optimasi, 134, 249
 - trade-off, 5
- klasifikasi, 54, 122
- kondisi batas, 256
- konektivitas jaringan, 229
- konsistensi
 - dan reproduksibilitas, 134
- konteks
 - input tak terbatas, 16
 - jendela, 15, 227
 - Pembuatan Konten Kontekstual, 189, 201, 202
 - pengambilan keputusan kontekstual, 227

- Saran Isian Kontekstual, 203
- konten
 - Kategorisasi Konten, 114
 - penyaringan, 27
- konten buatan pengguna, 113
- Kuantisasi, 28
- kustomisasi, 27
- language
 - models, 67
- Large Language Model (LLM), 29, 136, 145, 148, 169
- latensi, 28
- Latent Dirichlet Allocation, 124
- layanan atau API eksternal, 128
- lingkungan pengembangan lokal, 157
- lingkungan staging, 262
- Llama, 13
- Llama 2-70B, 51
- Llama 3 70B, 11
- Llama 3 8B, 11
- logika pemutus sirkuit, 164
- Louvre, 42
- Managed Streaming for Apache Kafka, 41
- manajemen lalu lintas, 33
- manajemen pengetahuan, 32
- Markdown, 148
- masalah kegunaan, 218
- mekanisme retry, 112
- mekanisme rollback, 262
- Memorial Sloan Kettering Cancer Center, 41
- mempersempit jalur, 38, 39
- mengintegrasikan LLM, 190
- Mercury (dewa Romawi), 45
- Mercury (planet), 45
- Mercury (unsur kimia), 45
- MessagePack, 255
- Meta, 24
- metode finalize, 158, 160, 161
- Metropolitan Museum of Art, 42
- Mistral, 25
 - 7B, 11
 - 7B Instruct, 17, 206
- Mixtral
 - 8x22B, 11
 - 8x7B, 57
- Model Bahasa Besar (LLM), 1, 3, 16, 18, 68, 70, 73, 77, 79, 89, 113, 122, 125, 126, 142, 146, 166, 189, 200, 206, 211, 234
 - lanskap, 27
- model berbasis pengambilan, 7
- model dasar, 55
- model grafis, 43
- model probabilistik, 43
- Moderator Konten Cerdas, 235
- modularitas, 90
- Multi-Agen
 - Pemecah Masalah, 31
- Multimodal
 - model, 20
 - model bahasa, 21
- Multitude of Workers, 121, 168

- nada emosional, 146
- Naive Bayes, 123
- New Relic, 253

- Ollama, 25
- Olympia, 34, 64, 130, 144, 153, 169
- OpenAI, 4, 23, 39, 75
- OpenRouter, 27, 28, 153, 253
- OPT model, 24
- optimasi
 - kinerja, 198
- optimistic locking, 112
- orkestrasi alur kerja cerdas, 223, 231, 252
- output verification, 256

- parafrase, 54
- parameter
 - efek, 130
 - Jumlah Parameter, 28
 - masukan, 130
 - rentang, 11
- pelacakan metrik kunci, 246
- pelatihan instruksi
 - model yang dilatih dengan instruksi,
 - 50, 53
- pemanggilan alat, 155
- pemantauan
 - dan pencatatan, 112, 248
 - dan peringatan, 229
 - metrik, 250
- Pemantauan Risiko Berkelanjutan, 105
- Pembangkitan Tertambah Pengambilan (RAG), 38
- pembangunan narasi, 20
- Pembatasan Respons, 179
- Pembelajaran Satu Contoh, 62
- pembelajaran tanpa contoh, 60, 61
- pembelajaran tanpa pengawasan, 4
- Pembersihan Teks, 114
- pembuatan data sintetis, 54
- pembuatan UI dinamis, 190
- pemodelan auto-regresif, 43
- pemrograman fungsional, 94
- pemrosesan aliran, 164
 - logika, 160
- pemrosesan asinkron, 251
- pemrosesan batch, 252
- pemrosesan stream, 152, 158
- pemungutan suara mayoritas, 119
- penalti pengulangan, 52
- Penalty Kehadiran, 49
- penandaan gaya markup, 72
- penangan stream, 153
- penanganan eksepsi, 231
- penanganan pengecualian, 228
- pencarian dan perbaikan bug, 227
- pencatatan audit, 108
- pencatatan terperinci, 250
- pencatatan terstruktur, 250
- pencocokan pola, 154
- penemuan medis, 103
- penerjemahan, 17
- pengalaman pengguna, 196
- pengambilan
 - keputusan kasus, 135

- pengelompokan dokumen, 122
- pengembangan aplikasi, 223
- penggunaan alat, 125, 150
- Pengkodean Pasangan Byte (BPE), 13
- pengujian end-to-end, 260, 261
- pengujian integrasi, 257
- pengujian pengguna dan umpan balik, 199
- Pengumpulan Riwayat Medis, 103
- pengungkapan progresif, 209
- Penilaian dan Stratifikasi Gejala, 103
- penugasan tiket, 242
- penulisan kreatif, 34, 54
- penyaringan berbasis konten, 94
- penyaringan kolaboratif, 94
- penyedia hosting model open source, 207
- penyelesaian berkinerja tinggi, 26
- penyempurnaan iteratif, 77
- penyetelan instruksi, 10
- penyimpanan cache, 252
- perangkat keras, 28
- perangkuman, 54
- perantaraan pekerja AI, 113
- percakapan
 - perulangan, 159
 - putaran, 161
 - transkrip, 161
- perencanaan tanggap darurat, 33
- perilaku deterministik, 59
- peritel daring, 207
- Perplexity (Provider), 12
- personalisasi, 190, 220, 225
 - Formulir yang Dipersonalisasi, 202
- personalization
 - Personalized Microcopy, 208
- Perutean Tugas Dinamis, 226
- pesan pemicu, 106
- pessimistic locking, 112
- pola historis, 227
- pola utama, 226
- prediksi, 5
- prinsip hak akses minimal, 73
- Process Manager, 106, 109
 - Enterprise Integration, 231
- Produktivitas, 192
- prompt
 - desain, 59, 69
 - Distilasi Prompt, 75
 - penyempurnaan, 70
 - Penyulingan Prompt, 80
 - perangkaian, 60
 - perantaraan, 73
 - Prompt Distillation, 252
 - Prompt Object, 76
 - rekayasa, 61, 68
 - Templat Prompt, 61
- prompts
 - engineering, 41, 45, 46, 57, 66, 217
 - Prompt Distillation, 46
 - Prompt Template, 207
- properti ACID, 112
- proses distilasi, 78
- Protocol Buffers, 255
- psikologi pengguna, 217
- pustaka Capybara, 261

- PyTorch, 25
- Qwen2 70B, 11
- Rails, 197
- Railway Oriented Programming (ROP), 97
- Raix, 232
 - pustaka, 99
- Rangkaian Pemikiran (CoT), 140
- ranker, 35
- rantai pasok
 - optimasi, 33
- regresi linear, 43
- Rekomendasi Produk, 94
- rekomendasi produk yang dipersonalisasi,
 - 94
- Response Fencing, 207
- retensi dan rotasi log, 250
- Retrieval Augmented Generation (RAG),
 - 46, 81
- RSpec, 256, 258, 261
- ruang laten, 40, 42
- Ruby, 95, 96, 115, 165, 261
- Ruby on Rails, 1, 113, 231, 239
- Rudall, Alex, 23
- Rust (Bahasa Pemrograman), 118
- Rust (Programming Language), 95
- Scout, 253
- server-sent events (SSE), 152
- sistem publikasi-langganan, 110
- sistem tanya-jawab, 7
- skalabilitas, 225, 251
- smartphones, 221
- stateless, 159
- strategi fallback, 112
- strategi motivasi, 215
- strategi segmentasi dan penargetan, 196
- Stratifikasi Risiko, 105
- Stripe, 131
- Structured IO, 207
- Support Vector Machines (SVM), 123
- syntax errors, 133
- system directive, 101
- T5, 24
- tablets, 221
- tantangan konseptual dan praktis, 201
- tanya jawab tertutup dan terbuka, 54
- Temperatur, 55
- teori pikiran, 40
- terjemahan, 198
- throughput, 28
- Together.ai, 26
- token, 6, 13
- tokenisasi, 13
- Top-k sampling, 48
- Top-p (nucleus) sampling, 48
- tragedy of the commons, 193
- tugas
 - terkait bahasa, 5
- tugas kompleks, 147
- UI adaptif, 210
- UI Generatif (GenUI), 212, 216, 220
- umpan balik

- Siklus Umpan Balik, 60
- Unicode-encodable language, 15
- Universal ID, 255
- User Interface (UI)
 - design, 221
- Ventriloquist, 179
- Verifikasi Asuransi, 104
- Waktu ke Token Pertama (TTFT), 28
- waktu pemrosesan, 112
- Wall, Larry, 3
- Wisper, 96, 108, 153, 160
- Wooley, Chad, 95
- XML, 136
- Yi-34B, 51