

Patterns of Application Development Using AI

Obie Fernandez

Foreword by Gregor Hohpe



한국어판

AI를 활용한 애플리케이션 개발 패턴 (한국어판)

Obie Fernandez

이 책은 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 할인 중입니다.

이 버전은 2025-01-23에 출간되었습니다.



이것은 [Leanpub](#) 책입니다. Leanpub은 작가들과 출판자들에게 Lean 출판 과정을 가지고 힘을 실어 줍니다. [Lean Publishing](#)은 가벼운 도구들과 많은 독자들의 피드백과 지지를 통해 일단 시작하면 앞으로 나아갈 힘을 구축하고, 당신이 제대로 된 책을 쓸때까지 계속해서 책을 출간하도록 하는 행위입니다.

© 2025 Obie Fernandez

이 책을 트위터에 보내기!

이 책에 대한 이야기를 [Twitter](#)에 올려 Obie Fernandez를 도와주세요!

이 책을 위한 권장 트위터 해시태그는 [#poaduai](#).

다른 사람들이 이 책에 대해 어떻게 말하고 있는지 확인하기 위해 다음 링크를 클릭하여 이 해시태그를 트위터에서 검색해보세요:

[#poaduai](#)

나의 용감무쌍한 여왕님, 나의 뮤즈, 나의 빛이자 사랑인 Victoria에게

또한 Obie Fernandez가 쓴 책들

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

차례

Gregor Hohpe의 서문	i
서문	ii
이 책에 대하여	iii
코드 예제에 대하여	iii
다루지 않는 내용	iii
이 책의 대상 독자	iii
공통 용어 구축하기	iii
참여하기	iii
감사의 말	iv
삽화는 어떻게 만들어졌나요?	iv
린 퍼블리싱에 대하여	iv
저자 소개	v
소개	1
소프트웨어 아키텍처에 대한 생각	2
대규모 언어 모델이란 무엇인가?	3
추론 이해하기	5
성능에 대한 고찰	25
다양한 LLM 모델 실험하기	26
복합 AI 시스템	27

Part 1: 기본적 접근법과 기술	34
경로 좁히기	35
잠재 공간: 이해할 수 없이 광대함	37
경로가 “좁혀지는” 방법	41
원본 모델과 명령어 투닝 모델 비교	44
프롬프트 엔지니어링	50
프롬프트 종류	66
파인튜닝은 어떤가요?	73
검색 증강 생성(RAG)	75
검색 증강 생성이란 무엇인가?	75
RAG는 어떻게 작동하는가?	75
왜 애플리케이션에서 RAG를 사용해야 하는가?	75
애플리케이션에 RAG 구현하기	75
명제 청킹	76
RAG의 실제 사례	77
지능형 쿼리 최적화 (IQO)	77
재순위화	77
RAG 평가 (RAGAs)	77
도전 과제와 미래 전망	79
다수의 작업자	82
독립적이고 재사용 가능한 컴포넌트로서의 AI 작업자	83
계정 관리	84
이커머스 애플리케이션	86
의료 분야 응용	96
프로세스 관리자로서의 AI 워커	98
AI 워커를 애플리케이션 아키텍처에 통합하기	102
AI 작업자의 구성 가능성과 오케스트레이션	105

전통적인 NLP와 LLM의 결합	115
도구 사용	118
도구 사용이란 무엇인가?	118
도구 사용의 잠재력	120
도구 사용 워크플로우	121
도구 사용을 위한 모범 사례	137
도구 구성 및 연결	142
향후 방향	144
스트림 처리	146
ReplyStream 구현하기	147
“대화 루프”	154
자동 연속	156
결론	158
자가 치유 데이터	160
실제 사례 연구: 손상된 JSON 수정	162
고려사항 및 금기사항	167
맥락 기반 콘텐츠 생성	181
개인화	182
생산성	183
빠른 반복과 실험	186
AI 기반 혼자화	188
사용자 테스트와 피드백의 중요성	189
생성형 UI	191
사용자 인터페이스를 위한 카피 생성	192
생성형 UI 정의하기	203
예시	205

결과 중심 디자인으로의 전환	207
과제와 고려사항	208
미래 전망과 기회	210
지능형 워크플로우 오케스트레이션	213
비즈니스 필요성	214
주요 이점	214
주요 패턴	215
예외 처리와 복구	217
실전에서의 지능형 워크플로우 오케스트레이션 구현	220
모니터링과 로깅	238
확장성 및 성능 고려사항	242
워크플로우의 테스트 및 검증	247
Part 2: 패턴들	255
프롬프트 엔지니어링	256
사고 연쇄	257
모드 전환	259
역할 할당	260
프롬프트 객체	261
프롬프트 템플릿	262
구조화된 입출력	263
프롬프트 채이닝	264
프롬프트 리라이터	265
응답 펜싱	266
쿼리 분석기	267
쿼리 재작성기	269
Ventriloquist	270

이산 컴포넌트	271
조건자	272
API 퍼사드	273
결과 해석기	276
가상 머신	277
명세와 테스트	277
휴면 인 더 루프 (HITL)	279
상위 수준 패턴	279
에스컬레이션	281
피드백 루프	282
수동적 정보 방사	283
협력적 의사결정 (CDM)	285
지속적 학습	286
윤리적 고려사항	286
기술적 발전과 미래 전망	287
지능형 오류 처리	288
전통적인 오류 처리 방식	288
맥락적 오류 진단	289
지능형 오류 보고	290
예측적 오류 방지	291
스마트 오류 복구	291
개인화된 오류 커뮤니케이션	292
적응형 오류 처리 워크플로우	293
품질 관리	294
Eval	295
안전장치	297
가드레일과 평가: 동전의 양면	298

용어집	299
용어집	299
Index	305

Gregor Hohpe의 서문

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

서문

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

이 책에 대하여

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

코드 예제에 대하여

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

다루지 않는 내용

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

이 책의 대상 독자

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

공통 용어 구축하기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

참여하기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

감사의 말

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

삽화는 어떻게 만들어졌나요?

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

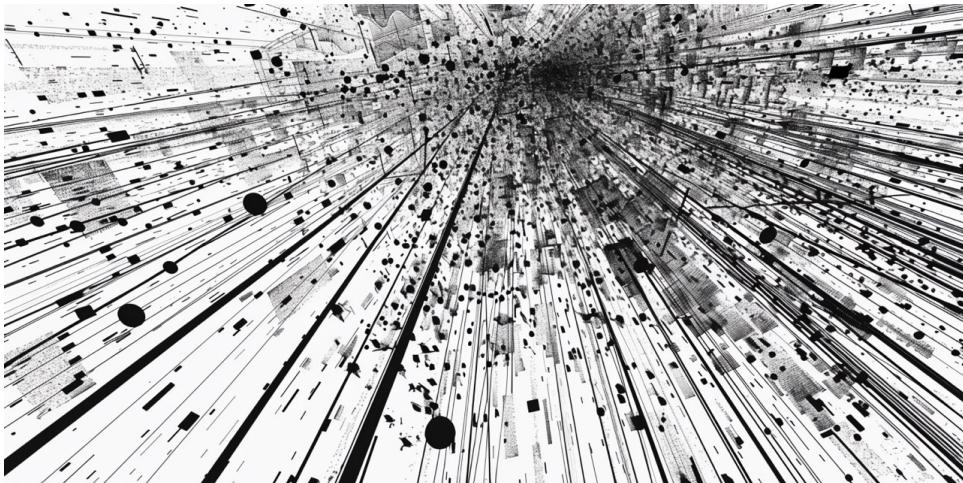
린 퍼블리싱에 대하여

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

저자 소개

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

소개



AI 대규모 언어 모델(LLM)을 프로그래밍 프로젝트에 통합하는 것을 빨리 시작하고 싶다면, 이후 장에서 소개되는 패턴과 코드 예제들을 바로 살펴보셔도 좋습니다. 하지만 이러한 패턴들의 힘과 잠재력을 충분히 이해하기 위해서는, 이들이 대표하는 더 넓은 맥락과 통합적 접근방식을 이해하는 데 잠시 시간을 할애할 가치가 있습니다.

이 패턴들은 단순히 고립된 기술들의 모음이 아니라, AI를 애플리케이션에 통합하기 위한 통일된 프레임워크입니다. 저는 Ruby on Rails를 사용하지만, 이러한 패턴들은 사실상 모든 프로그래밍 환경에서 작동할 것입니다. 이 패턴들은 데이터 관리와 성능 최적화부터 사용자 경험과 보안에 이르기까지 광범위한 문제들을 다루며, AI의 기능으로 전통적인 프로그래밍 관행을 향상시키기 위한 포괄적인 도구 모음을 제공합니다.

각 패턴 카테고리는 AI 컴포넌트를 애플리케이션에 통합할 때 발생하는 특정

과제나 기회를 다룹니다. 이러한 패턴들 간의 관계와 시너지를 이해함으로써, AI를 가장 효과적으로 적용할 수 있는 위치와 방법에 대해 정보에 기반한 결정을 내릴 수 있습니다.

패턴들은 결코 규범적인 해결책이 아니며 그렇게 취급되어서도 안 됩니다. 이들은 여러분의 고유한 애플리케이션의 특별한 요구사항과 제약조건에 맞춰 조정되어야 하는 적응형 구성 요소입니다. 이러한 패턴들의 성공적인 적용은 (소프트웨어 분야의 다른 모든 패턴들처럼) 문제 영역, 사용자 요구사항, 그리고 프로젝트의 전반적인 기술 아키텍처에 대한 깊은 이해를 필요로 합니다.

소프트웨어 아키텍처에 대한 생각

저는 1980년대에 프로그래밍을 시작했고 해커 문화에 몸담고 있었으며, 전문 소프트웨어 개발자가 된 후에도 해커의 사고방식을 잊지 않았습니다. 처음부터 저는 상아탑에 있는 소프트웨어 아키텍트들이 실제로 어떤 가치를 제공하는지에 대해 건전한 의구심을 가지고 있었습니다.

이 강력한 새로운 AI 기술이 가져온 변화에 대해 제가 개인적으로 매우 흥분하는 이유 중 하나는 우리가 소프트웨어 아키텍처라고 생각하는 결정들에 미치는 영향 때문입니다. 이는 소프트웨어 프로젝트를 설계하고 구현하는 “올바른” 방법에 대한 전통적인 개념에 도전합니다. 또한 AI 강화가 프로젝트의 어떤 부분이든 언제든지 변경하기 쉽게 만들고 있기 때문에, 아키텍처를 여전히 변경하기 어려운 시스템의 부분들로 주로 생각할 수 있는지에 대해서도 의문을 제기합니다.

아마도 우리는 소프트웨어 엔지니어링의 “포스트모던” 접근방식이 절정에 달하는 시기에 진입하고 있는 것 같습니다. 이 맥락에서 포스트모던이란 개발자들이 모든 코드 라인을 작성하고 유지보수하는 책임을 지는 전통적인 패러다임에서 근본적으로 벗어나는 것을 의미합니다. 대신, 데이터 조작, 복잡한 알고리즘, 심지어 애플리케이션 로직의 전체 부분을 서드파티 라이브러리와 외부 API에 위임하는 아이디어를 수용합니다. 이러한 포스트모던 전환은 애플리케이션을

처음부터 구축하는 전통적인 지혜에서 크게 벗어나는 것이며, 개발자들에게 개발 프로세스에서 자신의 역할을 재고하도록 도전합니다.

저는 Larry Wall과 그와 같은 다른 저명한 해커들의 가르침을 바탕으로, 좋은 프로그래머는 절대적으로 필요한 코드만 작성한다고 항상 믿어왔습니다. 작성하는 코드의 양을 최소화함으로써, 우리는 더 빠르게 움직일 수 있고, 버그가 발생할 수 있는 표면적을 줄이며, 유지보수를 단순화하고, 애플리케이션의 전반적인 신뢰성을 향상시킬 수 있습니다. 적은 양의 코드를 통해 다른 작업을 다른 서비스에 위임하면서 핵심 비즈니스 로직과 사용자 경험에 집중할 수 있습니다.

이제 AI 기반 시스템이 이전에는 사람이 작성한 코드만의 영역이었던 작업을 처리할 수 있게 되면서, 우리는 그 어느 때보다도 비즈니스 가치와 사용자 경험 창출에 더욱 집중하면서 더욱 생산적이고 민첩해질 수 있어야 합니다.

물론 프로젝트의 큰 부분을 AI 시스템에 위임하는 것에는 통제력 상실의 가능성과 강력한 모니터링 및 피드백 메커니즘의 필요성과 같은 트레이드오프가 있습니다. 그래서 AI가 어떻게 작동하는지에 대한 기본적인 이해를 포함한 새로운 기술과 지식이 필요합니다.

대규모 언어 모델이란 무엇인가?

대규모 언어 모델(LLM)은 2020년 OpenAI가 GPT-3를 출시한 이후 최근 몇 년간 큰 주목을 받고 있는 인공지능 모델의 한 유형입니다. LLM은 놀라운 정확도와 유창성으로 인간의 언어를 처리하고, 이해하며, 생성하도록 설계되었습니다. 이 섹션에서는 LLM이 어떻게 작동하는지, 그리고 왜 지능형 시스템 구성 요소를 구축하는 데 적합한지 간단히 살펴보겠습니다.

핵심적으로, LLM은 딥러닝 알고리즘, 특히 신경망을 기반으로 합니다. 이러한 네트워크는 정보를 처리하고 전달하는 상호 연결된 노드 또는 뉴런으로 구성됩니다. LLM에서 선호되는 아키텍처는 주로 트랜스포머 모델로, 텍스트와 같은 순차적 데이터를 처리하는 데 매우 효과적임이 입증되었습니다.

트랜스포머 모델은 어텐션 메커니즘을 기반으로 하며 주로 자연어 처리와 같은 순차적 데이터를 다루는 작업에 사용됩니다. 트랜스포머는 입력 데이터를 순차적으로 처리하는 대신 한 번에 처리하므로 장거리 의존성을 더 효과적으로 포착할 수 있습니다. 이들은 입력 데이터의 문맥과 관계를 이해하기 위해 모델이 다양한 부분에 집중할 수 있도록 돋는 어텐션 메커니즘 계층들을 가지고 있습니다.

대규모 언어 모델의 학습 과정에는 책, 기사, 웹사이트, 코드 저장소와 같은 방대한 양의 텍스트 데이터를 모델에 노출시키는 것이 포함됩니다. 학습 중에 모델은 텍스트 내의 패턴, 관계, 구조를 인식하는 법을 배웁니다. 문법 규칙, 단어 연관성, 문맥적 의미와 같은 언어의 통계적 특성을 포착합니다.

대규모 언어 모델 학습에 사용되는 주요 기술 중 하나는 비지도 학습입니다. 이는 모델이 명시적인 레이블이나 지도 없이 데이터로부터 학습한다는 것을 의미합니다. 학습 데이터에서 단어와 구문의 공동 출현을 분석함으로써 스스로 패턴과 표현을 발견합니다. 이를 통해 대규모 언어 모델은 언어와 그 복잡성에 대한 깊은 이해를 발전시킬 수 있습니다.

대규모 언어 모델의 또 다른 중요한 측면은 문맥을 다룰 수 있는 능력입니다. 텍스트를 처리할 때, 대규모 언어 모델은 개별 단어뿐만 아니라 주변 문맥도 고려합니다. 텍스트의 의미와 의도를 이해하기 위해 이전 단어, 문장, 심지어 단락까지도 고려합니다. 이러한 문맥적 이해를 통해 대규모 언어 모델은 일관성 있고 관련성 높은 응답을 생성할 수 있습니다. 특정 대규모 언어 모델의 능력을 평가하는 주요 방법 중 하나는 응답을 생성하기 위해 고려할 수 있는 문맥의 크기를 검토하는 것입니다.

학습이 완료되면, 대규모 언어 모델은 다양한 언어 관련 작업에 사용될 수 있습니다. 이들은 인간과 같은 텍스트를 생성하고, 질문에 답하고, 문서를 요약하고, 언어를 번역하고, 심지어 코드도 작성할 수 있습니다. 대규모 언어 모델의 다재다능함은 사용자와 상호작용하고, 텍스트 데이터를 처리하고 분석하며, 의미 있는 출력을 생성할 수 있는 지능형 시스템 구성 요소를 구축하는 데 가치가 있습니다.

애플리케이션 아키텍처에 대규모 언어 모델을 통합함으로써, 사용자 입력을 이해하고 처리하며, 동적 콘텐츠를 생성하고, 지능적인 추천이나 행동을 제공하는 AI 구성 요소를 만들 수 있습니다. 하지만 대규모 언어 모델을 다루기 위해서는 리소스 요구사항과 성능 트레이드오프를 신중히 고려해야 합니다. 대규모 언어 모델은 계산 집약적이며 작동하기 위해 상당한 처리 능력과 메모리 (다시 말해, 비용)가 필요할 수 있습니다. 대부분의 우리는 대규모 언어 모델을 애플리케이션에 통합하는 데 따르는 비용 영향을 평가하고 그에 따라 행동해야 할 것입니다.

추론 이해하기

추론은 모델이 새롭고 보지 못한 데이터를 기반으로 예측이나 출력을 생성하는 과정을 말합니다. 이는 학습된 모델이 사용자 입력에 대응하여 텍스트, 이미지 또는 기타 콘텐츠를 생성하거나 결정을 내리는 단계입니다.

학습 단계에서 AI 모델은 예측의 오류를 최소화하기 위해 매개변수를 조정하면서 대규모 데이터셋으로부터 학습합니다. 학습이 완료되면, 모델은 새로운 데이터에 학습한 내용을 적용할 수 있습니다. 추론은 모델이 학습한 패턴과 지식을 사용하여 출력을 생성하는 방법입니다.

대규모 언어 모델의 경우, 추론은 프롬프트나 입력 텍스트를 받아 일관성 있고 문맥상 관련성 있는 응답을 토큰 스트림으로 생성하는 것을 포함합니다(이에 대해서는 곧 이야기하겠습니다). 이는 질문에 답하기, 문장 완성하기, 이야기 생성하기, 텍스트 번역하기 등 다양한 작업이 될 수 있습니다.



여러분과 제가 생각하는 방식과는 대조적으로, AI 모델의 “사고”는 추론을 통해 하나의 상태가 없는 작업으로 이루어집니다. 즉, 그것의 사고는 생성 과정으로 제한됩니다. 마치 제가 여러분에게 질문을 하고 “의식의 흐름” 스타일로만 응답을 받아들이는 것처럼, 말 그대로 소리 내어 생각해야 합니다.

대규모 언어 모델은 다양한 크기와 특성으로 제공됩니다

실질적으로 모든 인기 있는 대규모 언어 모델(LLM)이 동일한 핵심 트랜스포머 아키텍처를 기반으로 하고 거대한 텍스트 데이터셋으로 학습되지만, 다양한 크기로 제공되며 서로 다른 목적을 위해 미세 조정됩니다. 신경망의 매개변수 수로 측정되는 대규모 언어 모델의 크기는 그 능력에 큰 영향을 미칩니다. 1~2 조 개의 매개변수를 자랑한다고 알려진 GPT-4와 같이 더 많은 매개변수를 가진 대형 모델들은 일반적으로 더 작은 모델들보다 더 많은 지식을 가지고 있고 더 유능합니다. 하지만 더 큰 모델들은 실행하는 데 더 많은 컴퓨팅 파워가 필요하며, 이는 API 호출을 통해 사용할 때 더 높은 비용으로 이어집니다.

대규모 언어 모델을 더 실용적이고 특정 사용 사례에 맞게 조정하기 위해, 기본 모델들은 종종 더 목표가 명확한 데이터셋으로 미세 조정됩니다. 예를 들어, 대규모 언어 모델은 대화형 AI를 위해 대규모 대화 말뭉치로 학습될 수 있습니다. 다른 모델들은 프로그래밍 지식을 부여하기 위해 [코드로 학습됩니다](#). 심지어 [사용자와의 룰플레이 스타일 상호작용을 위해 특별히 학습된](#) 모델들도 있습니다!

검색 vs 생성형 모델

대규모 언어 모델(LLM)의 세계에서는 응답을 생성하는 두 가지 주요 접근 방식이 있습니다: 검색 기반 모델과 생성형 모델입니다. 각 접근 방식에는 고유한 장단점이 있으며, 이들의 차이점을 이해하면 특정 사용 사례에 맞는 적절한 모델을 선택하는 데 도움이 됩니다.

검색 기반 모델

검색 기반 모델, 즉 정보 검색 모델은 기존의 방대한 텍스트 데이터베이스를 검색하고 입력 쿼리를 기반으로 가장 관련성이 높은 구절을 선택하여 응답을 생성합니다. 이러한 모델들은 새로운 텍스트를 처음부터 생성하지 않고, 대신 데이터베이스에서 발췌한 내용들을 조합하여 일관된 응답을 형성합니다.

검색 기반 모델의 주요 장점 중 하나는 사실에 근거하고 최신 정보를 제공할 수 있다는 점입니다. 이들은 선별된 텍스트 데이터베이스에 의존하기 때문에, 신뢰할 수 있는 출처에서 관련 정보를 추출하여 사용자에게 제시할 수 있습니다. 이는 질의응답 시스템 또는 지식 베이스와 같이 정확하고 사실에 기반한 답변이 필요한 애플리케이션에 매우 적합합니다.

하지만 검색 기반 모델에는 몇 가지 한계가 있습니다. 이들은 검색하는 데이터 베이스의 품질에 따라 성능이 좌우되므로, 데이터베이스의 품질과 범위가 모델의 성능에 직접적인 영향을 미칩니다. 또한, 이러한 모델들은 데이터베이스에 있는 텍스트로 제한되기 때문에 일관성이 있고 자연스러운 응답을 생성하는 데 어려움을 겪을 수 있습니다.

이 책에서는 순수 검색 모델의 사용법은 다루지 않습니다.

생성형 모델

반면에 생성형 모델은 학습 과정에서 배운 패턴과 관계를 바탕으로 새로운 텍스트를 처음부터 만들어냅니다. 이러한 모델들은 언어에 대한 이해를 바탕으로 입력 프롬프트에 맞춤화된 새로운 응답을 생성합니다.

생성형 모델의 주요 강점은 창의적이고 일관성이 있으며 문맥에 적절한 텍스트를 생성할 수 있다는 점입니다. 이들은 열린 대화에 참여하고, 이야기를 만들며, 심지어 코드도 작성할 수 있습니다. 이는 챗봇, 콘텐츠 생성, 창의적 글쓰기 보조와 같이 더 개방적이고 동적인 상호작용이 필요한 애플리케이션에 이상적입니다.

하지만 생성형 모델은 선별된 사실 데이터베이스가 아닌 학습 중에 습득한 패턴에 의존하기 때문에 때때로 일관성이 없거나 사실과 다른 정보를 생성할 수 있습니다. 또한 편향이나 환각에 더 취약할 수 있어, 그럴듯하지만 반드시 사실이 아닌 텍스트를 생성할 수 있습니다.

생성형 LLM의 예로는 OpenAI의 GPT 시리즈(GPT-3, GPT-4)와 Anthropic의 Claude가 있습니다.

하이브리드 모델

여러 상업용 LLM들은 검색과 생성 접근 방식을 모두 결합한 하이브리드 모델을 사용합니다. 이러한 모델들은 데이터베이스에서 관련 정보를 찾기 위해 검색 기술을 사용하고, 그 다음 생성 기술을 사용하여 해당 정보를 일관된 응답으로 종합합니다.

하이브리드 모델은 검색 기반 모델의 사실적 정확성과 생성형 모델의 자연어 생성 능력을 결합하는 것을 목표로 합니다. 이들은 개방형 대화에 참여할 수 있는 능력을 유지하면서도 더 신뢰할 수 있고 최신의 정보를 제공할 수 있습니다.

검색 기반 모델과 생성형 모델 중 선택할 때는 애플리케이션의 특정 요구사항을 고려해야 합니다. 주요 목표가 정확하고 사실에 기반한 정보를 제공하는 것이라면, 검색 기반 모델이 최선의 선택일 수 있습니다. 애플리케이션이 더 개방적이고 창의적인 상호작용을 요구한다면, 생성형 모델이 더 적합할 수 있습니다. 하이브리드 모델은 두 접근 방식 사이의 균형을 제공하며, 사실적 정확성과 자연어 생성이 모두 필요한 애플리케이션에 좋은 선택이 될 수 있습니다.

궁극적으로 검색 기반 모델과 생성형 모델 사이의 선택은 특정 사용 사례와 정확성, 창의성, 유연성 사이의 절충에 달려 있습니다. 각 접근 방식의 강점과 한계를 이해함으로써 정보에 기반한 결정을 내릴 수 있습니다.

명령어 튜닝

명령어 튜닝은 대화형 학습의 하위 집합으로, 사람이 작성한 프롬프트와 응답을 사용하여 LLM이 지시사항을 더 잘 이해하고 따르도록 만듭니다. 다음은 사람이 작성한 프롬프트와 응답으로 명령어 튜닝을 통해 LLM이 지시사항을 더 잘 이해하고 따르게 되는 과정을 보여주는 간단한 예시입니다:

예시 1: 고객 지원

지시사항: “환불을 요청하는 고객에게 응답을 제공하세요.”

사람이 작성한 프롬프트: “최근 구매한 제품에 대해 환불을 요청하고 싶습니다. 제품이 제 기대에 미치지 못했습니다.”

사람이 작성한 응답: “제품이 기대에 미치지 못했다니 죄송합니다. 환불 처리를 위해 주문 번호와 반품 사유를 제공해 주시기 바랍니다. 요청을 검토한 후 2~3영업일 이내에 답변 드리겠습니다.”

예시 2: 교육 지원

지시사항: “10살 아이에게 광합성의 개념을 설명하세요.”

Human-Written Prompt: “식물이 어떻게 음식을 만드는지 설명해줄 수 있나요?”

Human-Written Response: “네! 식물은 광합성이라는 과정을 통해 음식을 만듭니다. 햇빛, 물, 이산화탄소를 이용해 음식과 산소를 만들어냅니다. 식물의 잎에는 엽록소라는 특별한 부분이 있어서 햇빛을 포착하는 데 도움을 줍니다.”

Example 3: 개인 비서

Instruction: “다음 주 화요일 오후 3시에 회의를 예약해주세요.”

Human-Written Prompt: “화요일 오후 3시에 회의를 잡아줄 수 있나요?”

Human-Written Response: “네, 다음 주 화요일 오후 3시로 회의를 예약했습니다. 다른 필요한 것이 있으신가요?”

그 결과로 다양한 크기와 전문성을 가진 LLM의 생태계가 만들어졌습니다. 1~70억 매개변수 범위의 더 작은 모델들은 효율적으로 실행하면서도 좋은 일반적인 언어 능력을 제공합니다.

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

300~700억 매개변수 정도의 중간 크기 모델들은 더 강력한 추론과 지시사항 따르기 능력을 제공합니다.

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

애플리케이션에 통합할 LLM을 선택할 때는 모델의 능력과 비용, 자연 시간, 컨텍스트 길이, 콘텐츠 필터링과 같은 실용적인 요소들 사이의 균형을 맞춰야 합니다. 더 작고 명령어 조정된 모델들은 보통 간단한 언어 작업에 가장 적합한 선택이며, 가장 큰 모델들은 복잡한 추론이나 분석에 필요할 수 있습니다. 모델의 학습 데이터도 중요한 고려사항인데, 이는 모델의 지식 컷오프 날짜를 결정하기 때문입니다.



Perplexity의 일부 모델들처럼 특정 모델들은 실시간 정보 소스에 연결되어 있어서 사실상 컷오프 날짜가 없습니다. 이러한 모델들에게 질문을 하면, 독립적으로 웹 검색을 수행하고 임의의 웹 페이지를 가져와 답변을 생성할 수 있습니다.

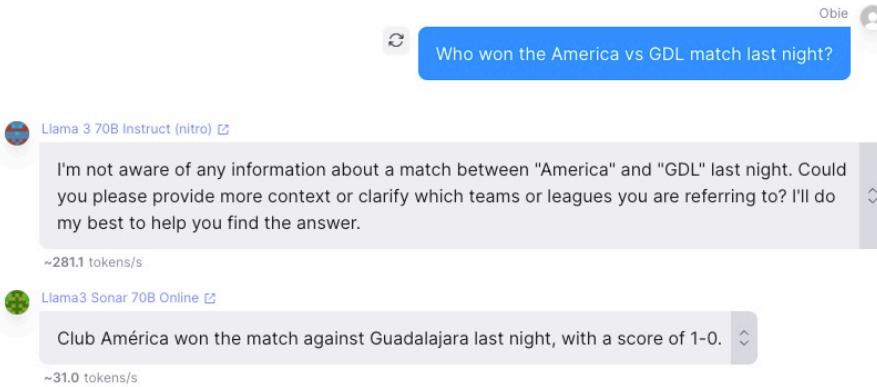


그림 1. 온라인 접속 가능 여부에 따른 Llama3의 차이

결국, 모든 상황에 적합한 단 하나의 LLM은 없습니다. 주어진 사용 사례에 맞는 적절한 모델을 선택하기 위해서는 모델 크기, 아키텍처, 학습의 차이를 이해하는 것이 핵심입니다. 실제로 어떤 모델이 해당 작업에 가장 좋은 성능을 제공하는지 알아내는 유일한 방법은 다양한 모델을 실험해보는 것입니다.

토큰화: 텍스트를 조각으로 나누기

거대 언어 모델이 텍스트를 처리하기 전에, 해당 텍스트를 토큰이라고 하는 더 작은 단위로 나눠야 합니다. 토큰은 개별 단어, 단어의 일부, 또는 단일 문자가 될 수 있습니다. 텍스트를 토큰으로 나누는 과정을 토큰화라고 하며, 이는 언어 모델을 위한 데이터 준비에서 중요한 단계입니다.

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

그림 2. 이 문장은 27개의 토큰을 포함합니다

다른 LLM들은 서로 다른 토큰화 전략을 사용하며, 이는 모델의 성능과 능력에 상당한 영향을 미칠 수 있습니다. LLM이 사용하는 일반적인 토크나이저에는

다음과 같은 것들이 있습니다:

- **GPT (바이트 쌍 인코딩):** GPT 토크나이저는 바이트 쌍 인코딩(BPE)이라 는 기술을 사용하여 텍스트를 하위 단어 단위로 나눕니다. BPE는 텍스트 코퍼스에서 가장 빈번한 바이트 쌍을 반복적으로 병합하여 하위 단어 토큰의 어휘를 형성합니다. 이를 통해 토크나이저는 드물거나 새로운 단 어를 더 일반적인 하위 단어 조각으로 분해하여 처리할 수 있습니다. GPT 토크나이저는 GPT-3와 GPT-4와 같은 모델에서 사용됩니다.
- **Llama (SentencePiece):** Llama 토크나이저는 비지도 텍스트 토크나 이저이자 디토크나이저인 SentencePiece 라이브러리를 사용합니다. SentencePiece는 입력 텍스트를 유니코드 문자의 시퀀스로 취급하고 학습 코퍼스를 기반으로 서브워드 어휘를 학습합니다. 유니코드로 인코딩할 수 있는 모든 언어를 처리할 수 있어 다국어 모델에 매우 적합합니다. Llama 토크나이저는 Meta의 Llama와 Alpaca와 같은 모델에서 사용됩니다.
- **SentencePiece (Unigram):** SentencePiece 토크나이저는 또한 서브워드 정규화 기술을 기반으로 하는 Unigram이라는 다른 알고리즘을 사용할 수 있습니다. Unigram 토큰화는 개별 서브워드 단위에 확률을 할당하는 유니그램 언어 모델을 기반으로 최적의 서브워드 어휘를 결정합니다. 이 접근 방식은 BPE와 비교하여 더 의미론적으로 유의미한 서브워드를 생성할 수 있습니다. Unigram을 사용하는 SentencePiece는 Google의 T5와 BERT와 같은 모델에서 사용됩니다.
- **Google Gemini (멀티모달 토큰화):** Google Gemini는 텍스트, 이미지, 오디오, 비디오, 코드를 포함한 다양한 데이터 유형을 처리할 수 있도록 설계된 토큰화 방식을 사용합니다. 이러한 멀티모달 기능을 통해 Gemini는 다양한 형태의 정보를 처리하고 통합할 수 있습니다. 특히 Google Gemini 1.5 Pro는 이전 모델보다 훨씬 큰 수백만 개의 토큰을 처리할 수 있는 컨텍스트 윈도우를 가지고 있습니다. 이러한 광범위한 컨텍스트 윈도우를 통해

모델은 더 큰 맥락을 처리할 수 있어 잠재적으로 더 정확한 응답을 생성할 수 있습니다. 그러나 Gemini의 토큰화 방식이 다른 모델들보다 문자당 하나의 토큰에 훨씬 더 가깝다는 점에 주목해야 합니다. 이는 Google의 가격이 토큰이 아닌 문자를 기준으로 책정되기 때문에, GPT와 같은 모델을 사용하는 데 익숙한 경우 Gemini 모델을 사용하는 실제 비용이 예상보다 훨씬 높을 수 있다는 것을 의미합니다.

토크나이저의 선택은 LLM의 여러 측면에 영향을 미칩니다:

- **어휘 크기:** 토크나이저는 모델이 인식하는 고유 토큰의 집합인 어휘의 크기를 결정합니다. 더 크고 세분화된 어휘는 모델이 더 넓은 범위의 단어와 구문을 처리하고 심지어 멀티모달(텍스트 이상의 것을 이해하고 생성할 수 있는 능력)이 될 수 있도록 도와주지만, 이는 또한 모델의 메모리 요구 사항과 계산 복잡성을 증가시킵니다.
- **희귀 단어와 알 수 없는 단어 처리:** BPE와 SentencePiece와 같이 서브워드 단위를 사용하는 토크나이저는 희귀하거나 알 수 없는 단어를 더 일반적인 서브워드 조각으로 분해할 수 있습니다. 이를 통해 모델은 이전에 보지 못한 단어의 의미를 포함된 서브워드를 기반으로 추측할 수 있습니다.
- **다국어 지원:** SentencePiece와 같이 유니코드로 인코딩 가능한 모든 언어를 처리할 수 있는 토크나이저는 여러 언어로 된 텍스트를 처리해야 하는 다국어 모델에 적합합니다.

특정 애플리케이션에 맞는 LLM을 선택할 때는 사용하는 토크나이저와 해당 작업의 특정 언어 처리 요구사항에 얼마나 잘 부합하는지 고려하는 것이 중요합니다. 토크나이저는 모델이 도메인별 용어, 희귀 단어, 다국어 텍스트를 처리하는 능력에 상당한 영향을 미칠 수 있습니다.

컨텍스트 크기: 언어 모델이 추론 중에 얼마나 많은 정보를 사용할 수 있을까요?

언어 모델을 논할 때, 컨텍스트 크기는 모델이 응답을 처리하거나 생성할 때 고려할 수 있는 텍스트의 양을 의미합니다. 이는 본질적으로 모델이 출력을 생성할 때 “기억”하고 사용할 수 있는 정보의 양(토큰으로 표현됨)을 측정한 것입니다. 언어 모델의 컨텍스트 크기는 그 기능과 효과적으로 수행할 수 있는 작업의 유형에 상당한 영향을 미칠 수 있습니다.

컨텍스트 크기란 무엇인가요?

기술적인 관점에서, 컨텍스트 크기는 언어 모델이 단일 입력 시퀀스에서 처리할 수 있는 토큰(단어 또는 단어 조각)의 수로 결정됩니다. 이는 종종 모델의 “어텐션 범위” 또는 “컨텍스트 윈도우”라고 불립니다. 컨텍스트 크기가 클수록 모델이 응답을 생성하거나 작업을 수행할 때 한 번에 고려할 수 있는 텍스트가 더 많아집니다.

다양한 언어 모델들은 수백 개의 토큰에서 수백만 개의 토큰에 이르기까지 서로 다른 컨텍스트 크기를 가지고 있습니다. 참고로, 일반적인 텍스트 단락은 약 100~150개의 토큰을 포함할 수 있으며, 전체 책은 수만 개에서 수십만 개의 토큰을 포함할 수 있습니다.

Transformer 기반 대규모 언어 모델(LLMs)을 무한히 긴 입력에 대해 제한된 메모리와 계산으로 확장하는 효율적인 방법에 대한 연구도 진행되고 있습니다.

문맥 크기가 중요한 이유는 무엇일까요?

언어 모델의 문맥 크기는 일관성 있고 문맥상 관련된 텍스트를 이해하고 생성하는 능력에 큰 영향을 미칩니다. 다음은 문맥 크기가 중요한 주요 이유들입니다:

1. 긴 형식의 콘텐츠 이해하기: 더 큰 문맥 크기를 가진 모델들은 기사, 보고서, 심지어 전체 책과 같은 더 긴 텍스트를 더 잘 이해하고 분석할 수 있습니다. 이는 문서 요약, 질문 답변, 콘텐츠 분석과 같은 작업에 매우 중요합니다.
2. 일관성 유지하기: 더 큰 문맥 창은 모델이 더 긴 출력 구간에서 일관성과 통일성을 유지할 수 있게 해줍니다. 이는 이야기 생성, 대화 시스템, 콘텐츠 제작과 같이 일관된 서사나 주제를 유지하는 것이 필수적인 작업에 중요합니다. 또한 구조화된 데이터를 생성하거나 변환하는 데 LLM을 사용할 때 절대적으로 중요합니다.
3. 장거리 의존성 파악하기: 일부 언어 작업은 텍스트에서 멀리 떨어져 있는 단어나 구절 간의 관계를 이해해야 합니다. 더 큰 문맥 크기를 가진 모델들은 이러한 장거리 의존성을 더 잘 파악할 수 있으며, 이는 감성 분석, 번역, 언어 이해와 같은 작업에 중요할 수 있습니다.
4. 복잡한 지시사항 처리하기: 언어 모델이 복잡한 다단계 지시사항을 따라야 하는 응용 프로그램에서, 더 큰 문맥 크기는 모델이 응답을 생성할 때 가장 최근의 몇 단어만이 아닌 전체 지시사항을 고려할 수 있게 해줍니다.

다양한 문맥 크기를 가진 언어 모델의 예시

다음은 서로 다른 문맥 크기를 가진 언어 모델의 몇 가지 예시입니다:

- OpenAI GPT-3.5 Turbo: 4,095 토큰
- Mistral 7B Instruct: 32,768 토큰
- Anthropic Claude v1: 100,000 토큰
- OpenAI GPT-4 Turbo: 128,000 토큰
- Anthropic Claude v2: 200,000 토큰
- Google Gemini Pro 1.5: 2.8M 토큰

보시다시피, OpenAI GPT-3.5 Turbo 모델의 약 4,000 토큰에서부터 Anthropic Claude v2 모델의 200,000 토큰까지 이러한 모델들 사이에는 광범위한 문맥 크기 차이가 있습니다. Google의 PaLM 2와 OpenAI의 GPT-4와 같은 일부 모델들은 더 긴 입력 시퀀스를 처리할 수 있는 더 큰 문맥 크기를 가진 변형("32k" 버전 등)을 제공합니다. 그리고 현재(2024년 4월) Google Gemini Pro는 거의 3배만 토큰을 자랑하고 있습니다!

특정 모델의 구현과 버전에 따라 문맥 크기가 달라질 수 있다는 점을 주목할 만합니다. 예를 들어, 원래의 OpenAI GPT-4 모델은 8,191 토큰의 문맥 크기를 가지고 있지만, Turbo와 4o와 같은 이후의 GPT-4 변형들은 128,000 토큰이라는 훨씬 더 큰 문맥 크기를 가지고 있습니다.

Sam Altman은 현재의 문맥 제한을 80년대 개인용 컴퓨터 프로그래머들이 다뤄야 했던 킬로바이트 단위의 작업 메모리에 비유했으며, 가까운 미래에는 “모든 개인 데이터”를 대규모 언어 모델의 문맥에 맞출 수 있을 것이라고 말했습니다.

적절한 문맥 크기 선택하기

특정 응용 프로그램에 맞는 언어 모델을 선택할 때, 해당 작업의 문맥 크기 요구사항을 고려하는 것이 중요합니다. 감성 분석이나 간단한 질문 답변과 같이 짧고 독립적인 텍스트를 다루는 작업의 경우, 더 작은 문맥 크기로도 충분할 수 있습니다. 하지만 더 길고 복잡한 텍스트를 이해하고 생성해야 하는 작업의 경우, 더 큰 문맥 크기가 필요할 것입니다.

더 큰 문맥 크기는 종종 모델이 응답을 생성할 때 더 많은 정보를 고려해야 하므로 더 높은 컴퓨팅 비용과 더 긴 처리 시간이 필요하다는 점을 주목할 필요가 있습니다. 따라서 응용 프로그램에 맞는 언어 모델을 선택할 때 문맥 크기와 성능 사이의 균형을 맞춰야 합니다.

가장 큰 문맥 크기를 가진 모델을 선택해서 가능한 한 많은 정보를 넣으면 되지 않을까요? 음, 성능 요소 외에도 또 다른 주요 고려사항은 비용입니다. 2024년 3월 기준으로 Google Gemini Pro 1.5를 전체 문맥으로 사용하는 단일 프롬프트-응답 주기에 거의 8달러(USD)가 듭니다. 그 정도의 비용이 정당화되는 사용 사례가 있다면 좋겠지만, 대부분의 응용 프로그램에서는 그것은 수십 배나 비싼 것입니다.

건초더미에서 바늘 찾기

건초더미에서 바늘을 찾는다는 개념은 오랫동안 대규모 데이터셋에서의 검색 과제를 설명하는 비유로 사용되어 왔습니다. LLM의 영역에서는 이 비유를 약간 변형해 봅시다. 방대한 텍스트(예: Paul Graham의 에세이 전집) 내에서 하나의 사실을 찾는 것이 아니라, 곳곳에 흩어져 있는 여러 사실들을 찾는 상황을 상상해보세요. 이는 마치 하나의 건초더미가 아닌 광활한 들판에서 여러 개의 바늘을 찾는 것과 비슷합니다. 여기서 중요한 점은 이 바늘들을 찾아내는 것뿐만 아니라, 이들을 하나의 일관된 실마리로 엮어내야 한다는 것입니다.

긴 맥락 속에 포함된 여러 사실들을 검색하고 추론하는 작업에서 LLM은 이중의 과제에 직면합니다. 첫째로, 검색 정확도의 문제가 있습니다—사실의 수가 증가할수록 자연스럽게 정확도가 떨어집니다. 이는 예상된 결과입니다. 결국, 방대한 텍스트에 걸쳐 여러 세부사항을 추적하는 것은 가장 정교한 모델에도 부담이 되기 때문입니다.

둘째로, 아마도 더 중요한 것은 이러한 사실들을 가지고 추론하는 과제입니다. 사실들을 찾아내는 것과 이를 일관된 서술이나 답변으로 종합하는 것은 전혀 다른 문제입니다. 여기서 진정한 시험이 시작됩니다. 추론 작업에서 LLM의 성능은 단순 검색 작업보다 더 크게 저하되는 경향이 있습니다. 이러한 성능 저하는 단순히 양의 문제가 아닙니다; 이는 맥락, 관련성, 추론이 복잡하게 얹혀 있는 문제입니다.

왜 이런 일이 발생할까요? LLM에서도 어느 정도 반영되는 인간의 인지 과정에서의 기억력과 주의력의 역학을 고려해보세요. 대량의 정보를 처리할 때, LLM은 인간처럼 새로운 정보를 흡수하면서 이전의 세부사항들을 놓칠 수 있습니다. 이는 특히 자동으로 텍스트의 이전 부분을 우선시하거나 재검토하도록 명시적으로 설계되지 않은 모델들에서 더욱 그렇습니다.

더욱이, LLM이 이렇게 검색된 사실들을 일관된 응답으로 엮어내는 능력은 서사 구축과 비슷합니다. 이는 단순한 정보 검색을 넘어서 깊은 이해와 맥락적 배치를 필요로 하며, 이는 현재의 AI에게 여전히 큰 도전과제로 남아있습니다.

그렇다면 이러한 기술들의 개발자이자 통합자로서 우리에게 이것이 의미하는 바는 무엇일까요? 우리는 LLM을 활용하여 복잡한 장문 작업을 처리하는 시스템을 설계할 때 이러한 한계를 예리하게 인식해야 합니다. 특정 조건에서 성능이 저하될 수 있다는 것을 이해하면 현실적인 기대치를 설정하고 더 나은 대체 메커니즘이나 보완 전략을 설계하는 데 도움이 됩니다.

모달리티: 텍스트를 넘어서

오늘날 대부분의 언어 모델이 텍스트의 처리와 생성에 중점을 두고 있지만, 이미지, 오디오, 비디오와 같은 여러 유형의 데이터를 기본적으로 입력하고 출력할 수 있는 멀티모달 모델로의 추세가 증가하고 있습니다. 이러한 멀티모달 모델들은 다양한 양식의 콘텐츠를 이해하고 생성할 수 있는 AI 기반 애플리케이션의 새로운 가능성을 열어줍니다.

모달리티란 무엇인가?

언어 모델의 맥락에서 모달리티란 모델이 처리하고 생성할 수 있는 다양한 유형의 데이터를 의미합니다. 가장 일반적인 모달리티는 책, 기사, 웹사이트, 소셜 미디어 게시물과 같은 다양한 형태의 문자 언어를 포함하는 텍스트입니다. 하지만 언어 모델에 점점 더 많이 통합되고 있는 다른 여러 모달리티들이 있습니다:

- **이미지:** 사진, 일러스트레이션, 다이어그램과 같은 시각적 데이터.
- **오디오:** 음성, 음악, 환경음과 같은 소리 데이터.
- **비디오:** 비디오 클립이나 영화와 같이 주로 오디오가 동반되는 동영상 데이터.

각 모달리티는 언어 모델에 고유한 과정과 기회를 제시합니다. 예를 들어, 이미지는 모델이 시각적 개념과 관계를 이해해야 하며, 오디오는 모델이 음성과 기타 소리를 처리하고 생성해야 합니다.

멀티모달 언어 모델

멀티모달 언어 모델은 단일 모델 내에서 여러 모달리티를 처리하도록 설계되었습니다. 이러한 모델들은 일반적으로 서로 다른 모달리티의 입력을 이해하고 출력 데이터를 생성할 수 있는 특수한 구성요소나 계층을 가지고 있습니다. 멀티모달 언어 모델의 주목할 만한 예시는 다음과 같습니다:

- **OpenAI의 GPT-4o:** GPT-4o는 텍스트 외에도 음성 오디오를 기본적으로 이해하고 처리할 수 있는 대규모 언어 모델입니다. 이 기능을 통해 GPT-4o는 구어를 전사하고, 오디오 입력에서 텍스트를 생성하며, 음성 쿼리에 기반한 응답을 제공하는 등의 작업을 수행할 수 있습니다.
- **OpenAI의 시각 입력이 가능한 GPT-4:** GPT-4는 텍스트와 이미지를 모두 처리할 수 있는 대규모 언어 모델입니다. 이미지를 입력으로 받으면 GPT-4는 이미지의 내용을 분석하고 시각적 정보를 설명하거나 이에 응답하는 텍스트를 생성할 수 있습니다.
- **Google의 Gemini:** Gemini는 텍스트, 이미지, 비디오를 처리할 수 있는 멀티모달 모델입니다. 교차 모달 이해와 생성을 가능하게 하는 통합 아키텍처를 사용하여 이미지 캡션 생성, 비디오 요약, 시각적 질의응답과 같은 작업을 수행할 수 있습니다.
- **DALL-E와 Stable Diffusion:** 전통적인 의미의 언어 모델은 아니지만, 이러한 모델들은 텍스트 설명으로부터 이미지를 생성하여 멀티모달 AI의

능력을 보여줍니다. 이는 서로 다른 양식 간의 변환이 가능한 모델의 잠재력을 보여줍니다.

멀티모달 모델의 이점과 응용

멀티모달 언어 모델은 다음과 같은 여러 이점을 제공하며 광범위한 응용이 가능합니다:

- **향상된 이해력:** 여러 양식의 정보를 처리함으로써, 이러한 모델들은 인간이 다양한 감각 입력을 통해 학습하는 것과 유사하게 세상에 대한 더 포괄적인 이해를 얻을 수 있습니다.
- **크로스모달 생성:** 멀티모달 모델은 한 양식의 입력을 기반으로 다른 양식의 콘텐츠를 생성할 수 있습니다. 예를 들어, 텍스트 설명으로부터 이미지를 만들거나 글로 작성된 기사로부터 비디오 요약을 생성하는 것이 가능합니다.
- **접근성:** 멀티모달 모델은 양식 간의 변환을 통해 정보를 더 접근하기 쉽게 만들 수 있습니다. 예를 들어, 시각 장애인을 위해 이미지의 텍스트 설명을 생성하거나 글로 작성된 콘텐츠의 오디오 버전을 만들 수 있습니다.
- **창의적 응용:** 멀티모달 모델은 텍스트 프롬프트를 기반으로 예술, 음악, 또는 비디오를 생성하는 등의 창의적 작업에 사용될 수 있어, 예술가와 콘텐츠 제작자들에게 새로운 가능성은 열어줍니다.

멀티모달 언어 모델이 계속 발전함에 따라, 여러 양식의 콘텐츠를 이해하고 생성할 수 있는 AI 기반 애플리케이션 개발에서 점점 더 중요한 역할을 할 것입니다. 이는 인간과 AI 시스템 간의 더 자연스럽고 직관적인 상호작용을 가능하게 하며, 창의적 표현과 지식 전파를 위한 새로운 가능성은 열어줄 것입니다.

제공업체 생태계

대규모 언어 모델(LLM)을 애플리케이션에 통합할 때, 선택할 수 있는 옵션이 점점 더 많아지고 있습니다. OpenAI, Anthropic, Google, Cohere 등 각 주요

LLM 제공업체는 자체적인 모델, API, 도구 생태계를 제공합니다. 적절한 제공업체를 선택하기 위해서는 가격, 성능, 콘텐츠 필터링, 데이터 프라이버시, 커스터마이제이션 옵션 등 다양한 요소를 고려해야 합니다.

OpenAI

OpenAI는 GPT 시리즈(GPT-3, GPT-4)가 다양한 애플리케이션에서 널리 사용되는 가장 잘 알려진 LLM 제공업체 중 하나입니다. OpenAI는 그들의 모델을 애플리케이션에 쉽게 통합할 수 있는 사용자 친화적인 API를 제공합니다. 그들은 기본 수준의 Ada 모델부터 강력한 Davinci 모델까지 다양한 능력과 가격대의 모델을 제공합니다.

OpenAI의 생태계에는 프롬프트를 실험하고 특정 사용 사례에 맞게 모델을 파인튜닝할 수 있는 OpenAI Playground와 같은 도구도 포함되어 있습니다. 또한 부적절하거나 유해한 콘텐츠 생성을 방지하는데 도움이 되는 콘텐츠 필터링 옵션을 제공합니다.

OpenAI의 모델을 직접 사용할 때는 Alex Rudall의 [ruby-openai](#) 라이브러리를 사용합니다.

Anthropic

Anthropic은 Claude 모델이 강력한 성능과 윤리적 고려사항으로 인기를 얻고 있는 LLM 분야의 또 다른 주요 업체입니다. Anthropic은 콘텐츠 필터링과 유해한 출력 방지에 중점을 두고 안전하고 책임감 있는 AI 시스템 개발에 초점을 맞추고 있습니다.

Anthropic의 생태계에는 모델을 애플리케이션에 통합할 수 있는 Claude API와 프롬프트 엔지니어링 및 파인튜닝을 위한 도구들이 포함되어 있습니다. 또한 더 최신의 사실적인 응답을 위해 웹 검색 기능을 통합한 Claude Instant 모델도 제공합니다.

Anthropic의 모델을 직접 사용할 때는 Alex Rudall의 [anthropic](#) 라이브러리를 사용합니다.

Google

Google은 Gemini, BERT, T5, PaLM을 포함한 여러 강력한 LLM을 개발했습니다. 이러한 모델들은 광범위한 자연어 처리 작업에서 강력한 성능을 보이는 것으로 알려져 있습니다. Google의 생태계에는 기계 학습 모델을 구축하고 훈련하기 위한 도구와 프레임워크를 제공하는 TensorFlow와 Keras 라이브러리가 포함되어 있습니다.

Google은 또한 클라우드에서 쉽게 모델을 배포하고 확장할 수 있는 Cloud AI Platform을 제공합니다. 감정 분석, 개체 인식, 번역과 같은 작업을 위한 다양한 사전 학습된 모델과 API를 제공합니다.

Meta

Meta, 이전의 Facebook은 LLaMA와 OPT와 같은 모델을 출시하며 대규모 언어 모델 개발에 깊이 투자하고 있습니다. 이러한 모델들은 다양한 언어 작업에서 뛰어난 성능을 보이며, 주로 오픈소스 채널을 통해 제공되어 Meta의 연구와 커뮤니티 협력에 대한 협신을 보여줍니다.

Meta의 생태계는 주로 동적 계산 능력과 유연성으로 혁신적인 AI 연구 개발을 촉진하는 오픈소스 기계 학습 라이브러리인 PyTorch를 중심으로 구축되어 있습니다.

Meta는 기술적인 제품 외에도 윤리적인 AI 개발에 큰 중점을 두고 있습니다. 강력한 콘텐츠 필터링을 구현하고 편향성 감소에 집중하며, AI 애플리케이션에서의 안전성과 책임성이라는 더 넓은 목표에 부합하고 있습니다.

Cohere

Cohere는 LLM 분야의 새로운 진입자로, 경쟁사들보다 LLM을 더 접근하기 쉽고 사용하기 쉽게 만드는 데 중점을 두고 있습니다. 그들의 생태계에는 Cohere API가 포함되어 있으며, 이를 통해 텍스트 생성, 분류, 요약과 같은 작업을 위한 다양한 사전 훈련된 모델에 접근할 수 있습니다.

Cohere는 또한 프롬프트 엔지니어링, 파인튜닝, 콘텐츠 필터링을 위한 도구들을 제공합니다. 암호화된 데이터 저장소와 접근 제어와 같은 기능을 통해 데이터 프라이버시와 보안을 강조합니다.

Ollama

Ollama는 자체 호스팅 플랫폼으로, 사용자가 외부 클라우드 서비스에 의존하지 않고 자신의 기기에서 로컬로 다양한 대규모 언어 모델(LLM)을 관리하고 배포할 수 있게 해줍니다. 이러한 설정은 데이터 프라이버시를 중요시하고 AI 운영을 사내에서 처리하고자 하는 사용자들에게 이상적입니다.

이 플랫폼은 Llama, Phi, Gemma, Mistral 등의 다양한 모델을 지원하며, 이들은 크기와 컴퓨팅 요구사항이 다양합니다. Ollama를 사용하면 `ollama run <model_name>`과 같은 간단한 명령어를 사용하여 명령줄에서 직접 이러한 모델들을 다운로드하고 실행할 수 있으며, macOS, Linux, Windows를 포함한 다양한 운영 체제에서 작동하도록 설계되었습니다.

원격 API를 사용하지 않고 오픈소스 모델을 애플리케이션에 통합하고자 하는 개발자들을 위해, Ollama는 컨테이너 관리 도구와 유사한 모델 수명주기 관리를 위한 CLI를 제공합니다. 또한 사용자 정의 구성과 프롬프트를 지원하여 특정 요구사항이나 사용 사례에 맞게 모델을 조정할 수 있는 높은 수준의 커스터마이제이션을 제공합니다.

Ollama는 명령줄 인터페이스와 AI 모델 관리 및 배포의 유연성 때문에 특히 기술에 능숙한 사용자와 개발자들에게 적합합니다. 이는 보안과 제어를 타협하지 않으면서 강력한 AI 기능이 필요한 기업과 개인들에게 강력한 도구가 됩니다.

멀티 모델 플랫폼

추가로, Together.ai와 Groq와 같이 다양한 오픈소스 모델을 호스팅하는 제공업체들이 있습니다. 이러한 플랫폼들은 유연성과 커스터마이제이션을 제공하여 특정 요구사항에 따라 오픈소스 모델을 실행하고, 경우에 따라서는 파인튜닝

까지 할 수 있게 해줍니다. 예를 들어, Together.ai는 다양한 오픈소스 LLM에 대한 접근을 제공하여 사용자가 다양한 모델과 구성을 실험할 수 있게 합니다. Groq는 이 책을 쓰는 시점에서 거의 마법처럼 보이는 초고성능 완성에 중점을 두고 있습니다.

LLM 제공업체 선택하기

LLM 제공업체를 선택할 때 고려해야 할 요소들:

- **가격책정:** 제공업체마다 사용량 기반 요금제부터 구독 기반 요금제까지 다양한 가격 모델을 제공합니다. 제공업체를 선택할 때 예상 사용량과 예산을 고려하는 것이 중요합니다.
- **성능:** LLM의 성능은 제공업체마다 크게 다를 수 있으므로, 결정을 내리기 전에 특정 사용 사례에 대해 벤치마크와 테스트를 수행하는 것이 중요합니다.
- **콘텐츠 필터링:** 애플리케이션에 따라 콘텐츠 필터링이 중요한 고려사항이 될 수 있습니다. 일부 제공업체는 다른 업체보다 더 강력한 콘텐츠 필터링 옵션을 제공합니다.
- **데이터 프라이버시:** 애플리케이션이 민감한 사용자 데이터를 다루는 경우, 강력한 데이터 프라이버시와 보안 관행을 가진 제공업체를 선택하는 것이 중요합니다.
- **커스터마이제이션:** 일부 제공업체는 특정 사용 사례를 위한 모델의 파인튜닝과 커스터마이징에 있어 더 많은 유연성을 제공합니다.

궁극적으로, LLM 제공업체 선택은 애플리케이션의 특정 요구사항과 제약조건에 따라 달라집니다. 가격, 성능, 데이터 프라이버시와 같은 요소들을 신중히 평가함으로써 귀하의 요구사항에 가장 잘 맞는 제공업체를 선택할 수 있습니다.

또한 LLM 환경이 새로운 제공업체와 모델이 정기적으로 등장하면서 지속적으로 발전하고 있다는 점을 주목할 가치가 있습니다. 최신 발전 상황을 계속 파악하고 새로운 옵션이 사용 가능해질 때 이를 탐색할 준비가 되어 있어야 합니다.

OpenRouter

이 책에서 저는 전적으로 [OpenRouter](#)를 API 제공업체로 사용할 것입니다. 그 이유는 간단합니다: 가장 인기 있는 상용 및 오픈소스 모델을 모두 이용할 수 있는 원스톱 솔이기 때문입니다. AI 코딩을 직접 해보고 싶으시다면, 제가 만든 [OpenRouter Ruby Library](#)로 시작하는 것이 가장 좋은 방법 중 하나입니다.

성능에 대한 고찰

애플리케이션에 언어 모델을 통합할 때 성능은 매우 중요한 고려사항입니다. 언어 모델의 성능은 자연 시간(응답을 생성하는 데 걸리는 시간)과 처리량(단위 시간당 처리할 수 있는 요청의 수)으로 측정될 수 있습니다.

첫 토큰 생성 시간(TTFT)은 또 다른 중요한 성능 지표로, 특히 챗봇과 실시간 상호작용이 필요한 애플리케이션에서 관련성이 높습니다. TTFT는 사용자의 요청이 수신된 순간부터 응답의 첫 단어(또는 토큰)가 생성되는 순간까지의 자연 시간을 측정합니다. 이 지표는 원활하고 매력적인 사용자 경험을 유지하는데 매우 중요합니다. 응답이 자연되면 사용자의 불만과 이탈로 이어질 수 있기 때문입니다.

이러한 성능 지표들은 사용자 경험과 애플리케이션의 확장성에 상당한 영향을 미칠 수 있습니다.

언어 모델의 성능에 영향을 미치는 요소들은 다음과 같습니다:

매개변수 수: 더 많은 매개변수를 가진 대형 모델은 일반적으로 더 많은 컴퓨팅 리소스를 필요로 하며, 작은 모델에 비해 자연 시간이 길어지고 처리량이 낮아질 수 있습니다.

하드웨어: 언어 모델의 성능은 실행되는 하드웨어에 따라 크게 달라질 수 있습니다. 클라우드 제공업체들은 머신 러닝 워크로드에 최적화된 GPU와 TPU 인스턴스를 제공하며, 이는 모델 추론을 크게 가속화할 수 있습니다.



OpenRouter의 장점 중 하나는 제공하는 많은 모델들에 대해 다양한 성능 프로필과 비용을 가진 클라우드 제공업체들 중에서 선택할 수 있다는 것입니다.

양자화: 양자화 기술은 가중치와 활성화를 더 낮은 정밀도의 데이터 타입으로 표현함으로써 모델의 메모리 사용량과 컴퓨팅 요구사항을 줄일 수 있습니다. 이를 통해 품질을 크게 희생하지 않고도 성능을 개선할 수 있습니다. 애플리케이션 개발자로서 다양한 양자화 수준에서 자체 모델을 훈련시키는 일에 관여하지는 않을 것이지만, 최소한 용어에 익숙해지는 것이 좋습니다.

배치 처리: 여러 요청을 동시에 배치로 처리하면 모델 로딩과 데이터 전송의 오버헤드를 분산시켜 처리량을 개선할 수 있습니다.

캐싱: 자주 사용되는 프롬프트나 입력 시퀀스의 결과를 캐싱하면 추론 요청의 수를 줄이고 전반적인 성능을 개선할 수 있습니다.

프로덕션 애플리케이션용 언어 모델을 선택할 때는 대표적인 워크로드와 하드웨어 구성에서 성능을 벤치마크하는 것이 중요합니다. 이를 통해 잠재적인 병목 현상을 식별하고 필요한 성능 목표를 달성을 할 수 있는지 확인할 수 있습니다.

또한 모델 성능과 비용, 유연성, 통합의 용이성 같은 다른 요소들 간의 균형을 고려하는 것도 중요합니다. 예를 들어, 실시간 응답이 필요한 애플리케이션의 경우 지역 시간이 낮은 더 작고 저렴한 모델을 사용하는 것이 바람직할 수 있으며, 배치 처리나 복잡한 추론 작업에는 더 크고 강력한 모델이 더 적합할 수 있습니다.

다양한 LLM 모델 실험하기

LLM을 선택하는 것은 거의 영구적인 결정이 아닙니다. 새롭고 개선된 모델들이 정기적으로 출시되므로, 시간이 지남에 따라 다른 언어 모델로 교체할 수 있는 모듈식으로 애플리케이션을 구축하는 것이 좋습니다. 프롬프트와 데이터셋은 최소한의 변경으로 여러 모델에서 재사용할 수 있는 경우가 많습니다. 이를

통해 애플리케이션을 완전히 재설계하지 않고도 언어 모델링의 최신 발전을 활용할 수 있습니다.



다양한 모델 선택지 간에 쉽게 전환할 수 있는 능력은 제가 OpenRouter를 좋아하는 또 다른 이유입니다.

새로운 언어 모델로 업그레이드할 때는 애플리케이션의 요구사항을 충족하는지 확인하기 위해 성능과 출력 품질을 철저히 테스트하고 검증하는 것이 중요합니다. 여기에는 도메인별 데이터로 모델을 재훈련하거나 미세 조정하는 것은 물론, 모델의 출력에 의존하는 모든 하위 구성 요소를 업데이트하는 작업이 포함될 수 있습니다.

성능과 모듈성을 고려하여 애플리케이션을 설계함으로써, 빠르게 진화하는 언어 모델링 기술 환경에 적응할 수 있는 확장 가능하고 효율적이며 미래 지향적인 시스템을 만들 수 있습니다.

복합 AI 시스템

우리의 소개를 마치기 전에, ChatGPT로 인해 생성형 AI에 대한 관심이 폭발적으로 증가한 2023년 이전에는 전통적인 AI 접근 방식이 일반적으로 단일 폐쇄 모델의 통합에 의존했다는 점을 언급할 가치가 있습니다. 이와 대조적으로, 복합 AI 시스템은 지능적 행동을 달성하기 위해 함께 작동하는 상호 연결된 구성 요소들의 복잡한 파이프라인을 활용합니다.

복합 AI 시스템의 핵심은 특정 작업이나 기능을 수행하도록 설계된 여러 모듈로 구성됩니다. 이러한 모듈에는 생성기, 검색기, 순위 결정기, 분류기 및 기타 다양한 전문화된 구성 요소가 포함될 수 있습니다. 전체 시스템을 더 작고 집중된 단위로 분해함으로써, 개발자들은 더 유연하고 확장 가능하며 유지보수가 용이한 AI 아키텍처를 만들 수 있습니다.

복합 AI 시스템의 주요 장점 중 하나는 서로 다른 AI 기술과 모델의 강점을 결합할 수 있다는 것입니다. 예를 들어, 시스템은 자연어 이해와 생성을 위해

대규모 언어 모델(LLM)을 사용하면서 정보 검색이나 규칙 기반 의사 결정을 위해 별도의 모델을 활용할 수 있습니다. 이러한 모듈식 접근 방식을 통해 획일적인 솔루션에 의존하는 대신 각각의 특정 작업에 가장 적합한 도구와 기술을 선택할 수 있습니다.

하지만 복합 AI 시스템을 구축하는 것은 독특한 과제들도 수반합니다. 특히 시스템 동작의 전반적인 일관성과 통일성을 보장하기 위해서는 강력한 테스트, 모니터링, 그리고 거버넌스 메커니즘이 필요합니다.



GPT-4와 같은 강력한 LLM의 등장으로 그 어느 때보다도 쉽게 복합 AI 시스템을 실험해볼 수 있게 되었습니다. 이는 이러한 고급 모델들이 자연어 이해 능력 외에도 분류, 순위 매기기, 생성과 같은 복합 시스템 내의 다양한 역할을 수행할 수 있기 때문입니다. 이러한 다재다능함은 개발자들이 복합 AI 아키텍처를 신속하게 프로토타입화하고 반복할 수 있게 해주어, 지능형 애플리케이션 개발에 새로운 가능성을 열어줍니다.

복합 AI 시스템을 위한 배포 패턴

복합 AI 시스템은 특정 요구사항과 사용 사례를 해결하기 위해 설계된 다양한 패턴을 사용하여 배포될 수 있습니다. 질의응답, 다중 에이전트/에이전틱 문제 해결사, 대화형 AI, 그리고 코파일럿이라는 네 가지 일반적인 배포 패턴을 살펴보겠습니다.

질의응답

질의응답(Q&A) 시스템은 단순한 검색 엔진 이상의 기능을 수행하기 위해 AI 모델의 이해 능력으로 강화된 정보 검색에 중점을 둡니다. **검색 강화 생성 (RAG)**을 사용하여 강력한 언어 모델과 외부 지식 소스를 결합함으로써, 질의 응답 시스템은 환각을 피하고 사용자 질문에 대해 정확하고 맥락에 맞는 응답을 제공합니다.

LLM 기반 Q&A 시스템의 주요 구성 요소는 다음과 같습니다:

- 질의 이해 및 재구성: 사용자 질의를 분석하고 기본 지식 소스에 더 잘 맞도록 재구성합니다.
- 지식 검색: 재구성된 질의를 기반으로 구조화되거나 비구조화된 데이터 소스에서 관련 정보를 검색합니다.
- 응답 생성: 검색된 지식과 언어 모델의 생성 능력을 통합하여 일관성 있고 유익한 응답을 생성합니다.

RAG 서브시스템은 고객 지원, 지식 관리, 또는 교육용 애플리케이션과 같이 정확하고 최신 정보를 제공하는 것이 중요한 Q&A 영역에서 특히 중요합니다.

다중 에이전트/에이전틱 문제 해결사

다중 에이전트, 즉 에이전틱 시스템은 복잡한 문제를 해결하기 위해 함께 작업하는 여러 자율 에이전트로 구성됩니다. 각 에이전트는 특정 역할, 기술 세트, 그리고 관련 도구나 정보 소스에 대한 접근 권한을 가지고 있습니다. 이러한 에이전트들은 협력하고 정보를 교환함으로써 단일 에이전트가 혼자서는 처리하기 어렵거나 불가능한 작업을 해결할 수 있습니다.

다중 에이전트 문제 해결사의 주요 원칙은 다음과 같습니다:

- 전문화: 각 에이전트는 고유한 능력과 지식을 활용하여 문제의 특정 측면에 집중합니다.
- 협력: 에이전트들은 메시지 전달이나 공유 메모리를 통해 공통의 목표를 달성하기 위해 소통하고 행동을 조율합니다.
- 적응성: 시스템은 개별 에이전트의 역할과 행동을 조정함으로써 변화하는 조건이나 요구사항에 적응할 수 있습니다.

다중 에이전트 시스템은 공급망 최적화, 교통 관리, 또는 비상 대응 계획과 같이 분산된 문제 해결이 필요한 애플리케이션에 적합합니다.

대화형 AI

대화형 AI 시스템은 사용자와 지능형 에이전트 간의 자연어 상호작용을 가능하게 합니다. 이러한 시스템은 매력적이고 개인화된 대화 경험을 제공하기 위해 자연어 이해, 대화 관리, 언어 생성 기능을 결합합니다.

대화형 AI 시스템의 주요 구성 요소는 다음과 같습니다:

- **의도 인식:** 질문하기, 요청하기, 감정 표현하기와 같은 사용자의 의도를 입력을 기반으로 식별합니다.
- **개체 추출:** 날짜, 위치, 제품명과 같은 관련 개체나 매개변수를 사용자의 입력에서 추출합니다.
- **대화 관리:** 대화의 상태를 유지하고, 사용자의 의도와 맥락에 기반하여 적절한 응답을 결정하며, 다중 턴 상호작용을 처리합니다.
- **응답 생성:** 언어 모델, 템플릿, 또는 검색 기반 방법을 사용하여 인간다운 응답을 생성합니다.

대화형 AI 시스템은 고객 서비스 챗봇, 가상 비서, 음성 제어 인터페이스에서 일반적으로 사용됩니다. 앞서 언급했듯이, 이 책의 대부분의 접근 방식, 패턴, 코드 예제는 [Olympia](#)라는 대규모 대화형 AI 시스템에 대한 제 작업에서 직접 추출한 것입니다.

코파일럿

코파일럿은 사용자의 생산성과 의사결정 능력을 향상시키기 위해 인간 사용자와 함께 작업하는 AI 기반 보조 시스템입니다. 이러한 시스템은 자연어 처리, 머신 러닝, 그리고 도메인별 전문 지식을 결합하여 지능적인 추천, 작업 자동화, 맥락에 맞는 지원을 제공합니다.

코파일럿의 주요 특징은 다음과 같습니다:

- **개인화:** 개별 사용자의 선호도, 작업 흐름, 의사소통 방식에 맞춰 적응합니다.

- **선제적 지원:** 명시적인 요청 없이도 사용자의 필요를 예측하고 관련된 제안이나 행동을 제공합니다.
- **지속적 학습:** 사용자 피드백, 상호작용, 데이터를 통해 시간이 지날수록 성능이 향상됩니다.

코파일럿은 소프트웨어 개발(예: 코드 완성 및 버그 감지), 창작 글쓰기(예: 콘텐츠 제안 및 편집), 데이터 분석(예: 인사이트 및 시각화 추천) 등 다양한 분야에서 점점 더 많이 사용되고 있습니다.

이러한 배포 패턴은 복합 AI 시스템의 다양성과 잠재력을 보여줍니다. 각 패턴의 특성과 사용 사례를 이해함으로써, 지능형 애플리케이션을 설계하고 구현할 때 정보에 기반한 결정을 내릴 수 있습니다. 이 책이 특별히 복합 AI 시스템의 구현에 관한 것은 아니지만, 기존의 전통적인 애플리케이션 개발 내에서 개별 AI 구성 요소를 통합하는 데에도 동일한 접근 방식과 패턴이 대부분 적용됩니다.

복합 AI 시스템의 역할

복합 AI 시스템은 특정 역할을 수행하도록 설계된 상호 연결된 모듈들의 기반 위에 구축됩니다. 이러한 모듈들은 함께 작동하여 지능적인 행동을 만들어내고 복잡한 문제를 해결합니다. 애플리케이션의 어느 부분을 개별 AI 구성 요소로 구현하거나 대체할 수 있을지 고려할 때 이러한 역할들을 숙지하는 것이 유용합니다.

생성기

생성기는 학습된 패턴이나 입력 프롬프트를 기반으로 새로운 데이터나 콘텐츠를 생성하는 역할을 합니다. AI 세계에는 다양한 종류의 생성기가 있지만, 이 책에서 다루는 언어 모델의 맥락에서 생성기는 사람과 같은 텍스트를 만들고, 불완전한 문장을 완성하거나, 사용자 질의에 대한 응답을 생성할 수 있습니다. 이들은 콘텐츠 생성, 대화 생성, 데이터 증강과 같은 작업에서 중요한 역할을 합니다.

검색기

검색기는 대규모 데이터셋이나 지식 베이스에서 관련 정보를 검색하고 추출하는 데 사용됩니다. 이들은 의미 검색, 키워드 매칭, 또는 벡터 유사도와 같은 기술을 사용하여 주어진 질의나 맥락에 기반하여 가장 적절한 데이터 포인트를 찾습니다. 검색기는 질문 답변, 사실 확인, 콘텐츠 추천과 같이 특정 정보에 대한 빠른 접근이 필요한 작업에 필수적입니다.

순위 결정기

순위 결정기는 특정 기준이나 관련성 점수를 기반으로 항목들의 순서를 정하거나 우선순위를 매기는 역할을 합니다. 각 항목에 가중치나 점수를 할당한 후 이에 따라 정렬합니다. 순위 결정기는 검색 엔진, 추천 시스템, 또는 사용자에게 가장 관련성 높은 결과를 제시하는 것이 중요한 모든 애플리케이션에서 일반적으로 사용됩니다.

분류기

분류기는 미리 정의된 클래스나 카테고리를 기반으로 데이터 포인트를 분류하거나 라벨링하는 데 사용됩니다. 이들은 라벨이 지정된 훈련 데이터로부터 학습한 후 새롭고 보지 못한 인스턴스의 클래스를 예측합니다. 분류기는 각 입력에 특정 카테고리를 할당하는 것이 목표인 감정 분석, 스팸 감지, 이미지 인식과 같은 작업에 기본이 됩니다.

도구 & 에이전트

이러한 핵심 역할 외에도, 복합 AI 시스템은 종종 기능성과 적응성을 향상시키기 위해 도구와 에이전트를 포함합니다:

- **도구:** 도구는 특정 작업이나 계산을 수행하는 개별 소프트웨어 구성 요소나 API입니다. 이들은 하위 작업을 수행하거나 추가 정보를 수집하기 위해

생성기나 검색기와 같은 다른 모듈에 의해 호출될 수 있습니다. 도구의 예로는 웹 검색 엔진, 계산기, 데이터 시각화 라이브러리 등이 있습니다.

- **에이전트:** 에이전트는 환경을 인식하고, 결정을 내리며, 특정 목표를 달성하기 위해 행동을 취할 수 있는 자율적인 개체입니다. 이들은 동적이거나 불확실한 조건에서 효과적으로 작동하기 위해 계획, 추론, 학습과 같은 다양한 AI 기술의 조합에 의존합니다. 에이전트는 복잡한 행동을 모델링하거나 복합 AI 시스템 내에서 여러 모듈의 행동을 조정하는 데 사용될 수 있습니다.

순수한 복합 AI 시스템에서, 이러한 구성 요소 간의 상호작용은 잘 정의된 인터페이스와 통신 프로토콜을 통해 조율됩니다. 데이터는 모듈 간에 흐르며, 한 구성 요소의 출력이 다른 구성 요소의 입력으로 사용됩니다. 이러한 모듈식 아키텍처는 개별 구성 요소를 전체 시스템에 영향을 주지 않고 업데이트, 교체 또는 확장할 수 있어 유연성, 확장성, 유지보수성을 제공합니다.

이러한 구성 요소와 그들의 상호작용의 힘을 활용함으로써, 복합 AI 시스템은 다양한 AI 기능의 조합이 필요한 복잡한 실제 문제를 해결할 수 있습니다. AI를 애플리케이션 개발에 통합하기 위한 접근 방식과 패턴을 탐구하면서, 복합 AI 시스템에서 사용되는 동일한 원칙과 기술을 지능적이고, 적응적이며, 사용자 중심적인 애플리케이션을 만드는 데 적용할 수 있다는 점을 기억하십시오.

파트 1의 다음 장들에서는 AI 구성 요소를 애플리케이션 개발 프로세스에 통합하기 위한 기본적인 접근 방식과 기술을 더 깊이 살펴볼 것입니다. 프롬프트 엔지니어링과 검색 증강 생성부터 자가 치유 데이터와 지능형 워크플로우 오키스트레이션에 이르기까지, 최첨단 AI 기반 애플리케이션을 구축하는 데 도움이 되는 다양한 패턴과 모범 사례를 다룰 것입니다.

Part 1: 기본적 접근법과 기술

이 책의 이 파트에서는 애플리케이션에 AI를 통합하는 다양한 방법을 소개합니다. 각 장에서는 경로 추적기와 검색 증강 생성과 같은 상위 수준의 개념부터 LLM 채팅 완성 API 위에 자체 추상화 계층을 프로그래밍하는 아이디어에 이르기까지 다양한 관련 접근법과 기술을 다룹니다.

이 파트의 목표는 [Part 2](#)에서 중점적으로 다룰 특정 구현 패턴으로 깊이 들어가기 전에, AI로 구현할 수 있는 다양한 종류의 동작을 이해하도록 돋는 것입니다.

Part 1의 접근법들은 제가 코드에서 사용했던 아이디어, 엔터프라이즈 애플리케이션 아키텍처와 통합의 고전적인 패턴, 그리고 비기술적인 비즈니스 이해관계자들을 포함한 다른 사람들에게 AI의 기능을 설명할 때 사용했던 비유들을 기반으로 합니다.

경로 좁히기



“경로 좁히기”는 AI를 주어진 작업에 집중시키는 것을 의미합니다. AI가 “멍청하게” 혹은 예상치 못한 방식으로 행동할 때마다 저는 이것을 주문처럼 되뇌입니다. 이 주문은 실패가 아마도 제 잘못이며, 제가 경로를 더 좁혀야 한다는 것을 상기시켜 줍니다.

경로를 좁혀야 하는 필요성은 거대 언어 모델에 포함된 방대한 양의 지식에서 비롯됩니다. 특히 OpenAI와 Anthropic과 같은 세계적 수준의 모델들은 문자 그대로 수조 개의 매개변수를 가지고 있습니다.

이렇게 광범위한 지식에 접근할 수 있다는 것은 분명 강력하며, 마음 이론과 인간적인 방식으로 추론하는 능력과 같은 창발적 행동을 만들어냅니다. 하지만, 이런 엄청난 양의 정보는 특정 프롬프트에 대해 정확하고 정밀한 응답을 생성할 때 어려움을 초래할 수 있습니다. 특히 이러한 프롬프트가 “일반적인” 소프트웨어 개발 및 알고리즘과 통합될 수 있는 결정론적 동작을 보여야 할 때 더욱 그렇습니다.

여러 요인들이 이러한 어려움을 야기합니다.

정보 과부하: 거대 언어 모델들은 다양한 도메인, 출처, 시대에 걸친 방대한 양의 데이터로 훈련됩니다. 이런 광범위한 지식은 모델이 다양한 주제에 참여하고 세상에 대한 폭넓은 이해를 바탕으로 응답을 생성할 수 있게 해줍니다. 하지만 특정 프롬프트에 직면했을 때, 모델은 관련 없거나 모순되거나 오래되거나 쓸모없는 정보를 걸러내는 데 어려움을 겪을 수 있어 초점이 흐려지거나 정확도가 떨어지는 응답을 할 수 있습니다. 여러분이 하고자 하는 일에 따라, 모델이 이용할 수 있는 모순된 정보의 엄청난 양은 여러분이 원하는 답변이나 행동을 제공하는 능력을 쉽게 압도할 수 있습니다.

맥락적 모호성: 방대한 잠재 공간의 지식을 고려할 때, 거대 언어 모델들은 프롬프트의 맥락을 이해하려 할 때 모호성에 직면할 수 있습니다. 적절한 좁힘이나 안내가 없다면, 모델은 여러분의 의도와 간접적으로 관련은 있지만 직접적으로 연관되지 않은 응답을 생성할 수 있습니다. 이러한 종류의 실패는 주제에서 벗어나거나, 일관성이 없거나, 명시된 요구사항을 충족시키지 못하는 응답으로 이어집니다. 이 경우 경로 좁히기는 맥락 명확화를 의미하며, 여러분이 제공하는 맥락이 모델로 하여금 기본 지식 중 가장 관련성 높은 정보에만 집중하도록 보장합니다.



참고: “프롬프트 앤지니어링”을 처음 시작할 때는 원하는 결과를 제대로 설명하지 않고 모델에게 작업을 요청하는 경우가 훨씬 많습니다. 모호하지 않게 설명하려면 연습이 필요합니다!

시간적 불일치: 언어 모델들은 서로 다른 시기에 생성된 데이터로 훈련되기

때문에, 오래되거나, 대체되었거나, 더 이상 정확하지 않은 지식을 보유할 수 있습니다. 예를 들어, 시사 문제, 과학적 발견, 기술적 진보에 대한 정보는 모델의 훈련 데이터가 수집된 이후로 발전했을 수 있습니다. 더 최근의 신뢰할 수 있는 출처를 우선시하도록 경로를 좁히지 않으면, 모델은 오래되거나 부정확한 정보를 바탕으로 응답을 생성할 수 있어 출력물에 부정확성과 불일치가 발생할 수 있습니다.

도메인별 뉘앙스: 서로 다른 도메인과 분야는 각자 고유한 용어, 관례, 지식 기반을 가지고 있습니다. 실질적으로 어떤 TLA(세 글자 약어)든 생각해보면, 대부분이 하나 이상의 의미를 가지고 있다는 것을 알 수 있습니다. 예를 들어, MSK는 Amazon의 Managed Streaming for Apache Kafka, Memorial Sloan Kettering Cancer Center, 또는 인체의 근골격계(MusculoSKeletal system)를 지칭할 수 있습니다.

프롬프트가 특정 도메인의 전문 지식을 요구할 때, 거대 언어 모델의 일반적인 지식만으로는 정확하고 뉘앙스가 살아있는 응답을 제공하기에 충분하지 않을 수 있습니다. 프롬프트 엔지니어링이나 검색 증강 생성을 통해 도메인별 정보에 집중하도록 경로를 좁히면, 모델이 특정 도메인의 요구사항과 기대에 더 잘 부합하는 응답을 생성할 수 있습니다.

잠재 공간: 이해할 수 없이 광대함

제가 언어 모델의 “잠재 공간”을 언급할 때, 이는 모델이 훈련 과정에서 학습한 지식과 정보의 광대하고 다차원적인 풍경을 의미합니다. 이는 모델의 신경망 내부에 있는 숨겨진 영역과 같으며, 여기에는 언어의 모든 패턴, 연관성, 표현이 저장되어 있습니다.

수많은 상호 연결된 노드로 가득 찬 광대한 미개척 영역을 탐험하는 것을 상상해 보세요. 각 노드는 모델이 학습한 정보 조각, 개념, 또는 관계를 나타냅니다. 이 공간을 탐색하다 보면, 일부 노드들은 더 가까이 있어 강한 연결이나 유사성을 나타내고, 다른 노드들은 더 멀리 떨어져 있어 더 약하거나 먼 관계를 나타냄을

발견하게 될 것입니다.

잠재 공간의 어려운 점은 그것이 믿을 수 없을 정도로 복잡하고 고차원적이라는 것입니다. 마치 우리의 물리적 우주처럼, 은하게 군집들과 그 사이의 상상할 수 없이 광활한 빈 공간이 있는 것처럼 생각해보세요.

수천 개의 차원을 포함하고 있기 때문에, 잠재 공간은 인간이 직접 관찰하거나 해석할 수 없습니다. 이는 모델이 내부적으로 언어를 처리하고 생성하는 데 사용하는 추상적인 표현입니다. 모델에 입력 프롬프트를 제공하면, 본질적으로 해당 프롬프트를 잠재 공간 내의 특정 위치에 매핑합니다. 그런 다음 모델은 해당 공간의 주변 정보와 연결을 사용하여 응답을 생성합니다.

문제는 모델이 학습 데이터에서 엄청난 양의 정보를 학습했지만, 주어진 작업에 모든 정보가 관련있거나 정확한 것은 아니라는 점입니다. 그래서 경로를 좁히는 것이 매우 중요해집니다. 프롬프트에 명확한 지시사항, 예시, 맥락을 제공함으로써, 본질적으로 모델이 잠재 공간 내에서 원하는 출력과 가장 관련있는 특정 영역에 집중하도록 안내하는 것입니다.

이를 다르게 생각해보면, 완전히 어두운 박물관에서 스포트라이트를 사용하는 것과 같습니다. 루브르 또는 메트로폴리탄 미술관을 방문해보셨다면, 제가 말하는 규모가 어느 정도인지 아실 것입니다. 잠재 공간은 수많은 물건과 세부사항으로 가득 찬 박물관과 같습니다. 여러분의 프롬프트는 스포트라이트와 같아서, 특정 영역을 비추고 모델의 주의를 가장 중요한 정보로 이끕니다. 이러한 안내가 없다면, 모델은 잠재 공간을 무작정 돌아다니며 관련 없거나 모순되는 정보를 주워 담을 수 있습니다.

언어 모델로 작업하고 프롬프트를 작성할 때, 잠재 공간의 개념을 염두에 두세요. 여러분의 목표는 이 방대한 지식 공간을 효과적으로 탐색하여, 모델이 작업에 가장 관련있고 정확한 정보를 향해 나아가도록 하는 것입니다. 경로를 좁히고 명확한 안내를 제공함으로써, 모델의 잠재 공간의 잠재력을 최대한 활용하여 높은 품질의 일관된 응답을 생성할 수 있습니다.

앞서 설명한 언어 모델과 그들이 탐색하는 잠재 공간이 다소 마법적이거나 추상적으로 들릴 수 있지만, 프롬프트가 주문이나 주술이 아니라는 점을 이해하는

것이 중요합니다. 언어 모델의 작동 방식은 선형대수와 확률 이론의 원리에 기반을 두고 있습니다.

핵심적으로, 언어 모델은 텍스트의 확률적 모델입니다. 마치 종 모양 곡선이 데이터의 통계적 모델인 것처럼요. 이들은 자기회귀 모델링이라는 과정을 통해 훈련되는데, 이는 모델이 이전에 나온 단어들을 기반으로 시퀀스에서 다음 단어의 확률을 예측하는 것을 학습하는 과정입니다. 훈련 중에 모델은 무작위 가중치로 시작하여 점차적으로 이를 조정하면서, 실제 학습 샘플과 유사한 텍스트에 더 높은 확률을 할당하게 됩니다.

하지만 언어 모델을 선형 회귀와 같은 단순한 통계 모델로 생각하는 것은 그들의 행동을 이해하는 데 최적의 직관을 제공하지 않습니다. 더 적절한 비유는 이들을 확률적 프로그램으로 생각하는 것입니다. 이는 무작위 변수를 조작할 수 있고 복잡한 통계적 관계를 표현할 수 있는 모델입니다.

확률적 프로그램은 그래프 모델로 표현될 수 있는데, 이는 모델 내의 변수들 간의 의존성과 관계를 시각적으로 이해할 수 있게 해줍니다. 이러한 관점은 GPT-4와 Claude와 같은 복잡한 텍스트 생성 모델의 작동 방식에 대한 귀중한 통찰을 제공할 수 있습니다.

Dohan 등의 “Language Model Cascades” 논문에서, 저자들은 확률적 프로그램이 어떻게 언어 모델에 적용될 수 있는지에 대해 자세히 다룹니다. 그들은 이 프레임워크가 어떻게 이러한 모델들의 행동을 이해하고 더 효과적인 프롬프팅 전략의 개발을 안내하는 데 사용될 수 있는지 보여줍니다.

이러한 확률적 관점에서 얻은 핵심 통찰 중 하나는 언어 모델이 본질적으로 원하는 문서가 존재하는 대체 우주로의 포털을 만든다는 것입니다. 모델은 모든 가능한 문서에 확률에 기반하여 가중치를 할당하여, 가능성의 공간을 가장 관련있는 것들에 집중하도록 효과적으로 좁힙니다.

이는 우리를 다시 “경로 좁히기”라는 중심 주제로 되돌립니다. 프롬프팅의 주요 목표는 확률적 모델을 조건화하여 예측의 질량을 집중시키는 것으로, 우리가 이끌어내고자 하는 특정 정보나 행동에 초점을 맞추는 것입니다. 신중하게 작성된 프롬프트를 제공함으로써, 우리는 모델이 잠재 공간을 더 효율적으로

탐색하고 더 관련성 있고 일관된 출력을 생성하도록 안내할 수 있습니다.

하지만 언어 모델이 궁극적으로 학습한 정보에 의해 제한된다는 점을 명심하는 것이 중요합니다. 기존 문서와 유사한 텍스트를 생성하거나 아이디어를 새로운 방식으로 결합할 수는 있지만, 완전히 새로운 정보를 무에서 창조할 수는 없습니다. 예를 들어, 암에 대한 치료법이 아직 발견되지 않고 학습 데이터에 문서화되지 않았다면, 모델에게 암 치료법을 제공하도록 기대할 수는 없습니다.

대신, 이 모델의 강점은 우리가 프롬프트로 제시하는 것과 유사한 정보를 찾아내고 종합하는 능력에 있습니다. 이러한 모델의 확률적 특성과 프롬프트가 출력을 조정하는 방식을 이해함으로써, 우리는 더욱 효과적으로 이들의 능력을 활용하여 가치 있는 통찰과 콘텐츠를 생성할 수 있습니다.

아래의 프롬프트들을 살펴보겠습니다. 첫 번째에서 “Mercury” 하나만으로는 행성을 가리킬 수도 있고, 원소를 가리킬 수도 있으며, 로마의 신을 가리킬 수도 있지만, 가장 확률이 높은 것은 행성입니다. 실제로 GPT-4는 수성은 태양계에서 가장 작고 가장 안쪽에 있는 행성입니다…로 시작하는 긴 응답을 제공합니다. 두 번째 프롬프트는 구체적으로 화학 원소를 지칭합니다. 세 번째는 속도와 신의 전령으로서의 역할로 알려진 로마 신화의 인물을 가리킵니다.

- 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

몇 개의 단어만 추가했을 뿐인데, AI의 반응이 완전히 달라졌습니다. 이 책의 뒷 부분에서 배우게 되겠지만, n-shot 프롬프팅, 구조화된 입력/출력, 사고 연쇄와

같은 정교한 프롬프트 엔지니어링 기법들은 모델의 출력을 조절하는 영리한 방법일 뿐입니다.

결국, 프롬프트 엔지니어링의 기술은 언어 모델의 지식이라는 광대한 확률적 풍경을 탐색하여, 우리가 찾는 특정 정보나 행동으로 가는 경로를 좁혀나가는 것에 관한 것입니다.

고급 수학에 능통한 독자들의 경우, 이러한 모델들을 확률론과 선형 대수학의 원리에 기반하여 이해하면 확실히 도움이 될 것입니다! 효과적인 출력 유도 전략을 개발하고자 하는 나머지 독자들을 위해서는, 좀 더 직관적인 접근 방식을 사용해보겠습니다.

경로가 “좁혀지는” 방법

너무 많은 지식이라는 이러한 과제를 해결하기 위해, 우리는 언어 모델의 생성 과정을 안내하고 가장 관련성 있고 정확한 정보에 집중하도록 돋는 기술들을 사용합니다.

다음은 권장 순서대로 나열한 가장 중요한 기술들입니다. 즉, 프롬프트 엔지니어링을 먼저 시도하고, 그 다음 RAG를 시도하며, 마지막으로 꼭 필요한 경우에만 파인튜닝을 사용해야 합니다.

프롬프트 엔지니어링 가장 기본적인 접근 방식은 모델의 응답 생성을 안내하기 위한 구체적인 지침, 제약 조건, 또는 예시를 포함하는 프롬프트를 만드는 것입니다. 이 장에서는 [다음 섹션](#)에서 프롬프트 엔지니어링의 기초를 다루며, 책의 제2부에서는 많은 구체적인 프롬프트 엔지니어링 패턴을 다룹니다. 이러한 패턴들에는 [프롬프트 종류](#)가 포함되는데, 이는 AI가 가장 관련성 있고 간결한 정보라고 판단하는 것을 추출하기 위해 프롬프트를 정제하고 최적화하는 기법입니다.

컨텍스트 증강. 프롬프트가 주어지는 시점에 외부 지식 베이스나 문서에서 관련 정보를 동적으로 검색하여 모델에 집중된 컨텍스트를 제공합니다. 인기 있는 컨텍스트 증강 기술에는 [검색 증강 생성\(RAG\)](#)이 포함됩니다. [Perplexity](#)가

제공하는 것과 같은 이른바 “온라인 모델”들은 실시간 인터넷 검색 결과로 컨텍스트를 증강할 수 있습니다.



LLM들이 강력하진 하지만, 여러분만의 고유한 데이터셋에 대해서는 학습되어 있지 않습니다. 이러한 데이터셋은 비공개이거나 여러분이 해결하려는 문제에 특화되어 있을 수 있습니다. 컨텍스트 증강 기술을 사용하면 LLM이 API 뒤에 있는 데이터, SQL 데이터베이스, 또는 PDF와 슬라이드 테크에 간접 데이터에 접근할 수 있게 됩니다.

파인튜닝 또는 도메인 적응 특정 작업이나 분야에 대한 지식과 생성 능력을 특화시키기 위해 도메인별 데이터셋으로 모델을 훈련시키는 것입니다.

템퍼러처 낮추기

템퍼러처는 트랜스포머 기반 언어 모델에서 생성되는 텍스트의 무작위성과 창의성을 제어하는 하이퍼파라미터입니다. 0에서 1 사이의 값을 가지며, 낮은 값은 출력력을 더 집중적이고 결정적으로 만들고, 높은 값은 더 다양하고 예측 불가능하게 만듭니다.

템퍼러처가 1로 설정되면, 언어 모델은 다음 토큰의 전체 확률 분포를 기반으로 텍스트를 생성하여 더 창의적이고 다양한 응답을 가능하게 합니다. 하지만 이로 인해 모델이 덜 관련성 있거나 덜 일관된 텍스트를 생성할 수도 있습니다.

반면에 템퍼러처가 0으로 설정되면, 언어 모델은 항상 가장 높은 확률을 가진 토큰을 선택하여 효과적으로 “경로를 좁힙니다.” 제가 사용하는 AI 구성 요소들은 거의 모두 0이나 그에 가까운 템퍼러처를 사용하는데, 이는 더 집중적이고 예측 가능한 응답을 얻을 수 있기 때문입니다. 모델이 지침을 따르게 하거나, 제공된 함수에 주의를 기울이게 하거나, 단순히 현재보다 더 정확하고 관련성 있는 응답이 필요할 때 절대적으로 유용합니다.

예를 들어, 사실적인 정보를 제공해야 하는 챗봇을 만들 때는 응답이 더 정확하고 주제에 맞도록 하기 위해 템퍼러처를 낮은 값으로 설정하고 싶을 것입니다.

반대로 창의적인 글쓰기 도우미를 만들 때는 더 다양하고 상상력 풍부한 출력을 장려하기 위해 템퍼러처를 높은 값으로 설정하고 싶을 것입니다.

하이퍼파라미터: 추론의 손잡이와 다이얼

언어 모델을 다룰 때 “하이퍼파라미터”라는 용어를 자주 접하게 될 것입니다. 추론의 맥락에서(즉, 응답을 생성하기 위해 모델을 사용할 때), 하이퍼파라미터는 모델의 행동과 출력을 제어하기 위해 조정할 수 있는 손잡이와 다이얼 같은 것입니다.

복잡한 기계의 설정을 조정하는 것처럼 생각해보세요. 온도를 조절하기 위해 손잡이를 돌리거나 작동 모드를 변경하기 위해 스위치를 전환하는 것처럼, 하이퍼파라미터를 통해 언어 모델이 텍스트를 처리하고 생성하는 방식을 세밀하게 조정할 수 있습니다.

추론 과정에서 자주 마주치게 될 하이퍼파라미터들은 다음과 같습니다:

- **Temperature(온도):** 앞서 언급했듯이, 이 매개변수는 생성된 텍스트의 무작위성과 창의성을 제어합니다. 높은 온도는 더 다양하고 예측할 수 없는 출력을 만들어내는 반면, 낮은 온도는 더 집중적이고 결정론적인 응답을 만들어냅니다.
- **Top-p (핵) 샘플링:** 이 매개변수는 누적 확률이 특정 임계값(p)을 초과하는 가장 작은 토큰 집합의 선택을 제어합니다. 이는 일관성을 유지하면서도 더 다양한 출력을 가능하게 합니다.
- **Top-k 샘플링:** 이 기법은 가장 가능성성이 높은 다음 k 개의 토큰을 선택하고 그들 사이에 확률 질량을 재분배합니다. 이는 모델이 낮은 확률이나 관련 없는 토큰을 생성하는 것을 방지하는 데 도움이 됩니다.
- **빈도와 존재 폐널티:** 이러한 매개변수들은 동일한 단어나 구문을 너무 자주 반복하는 것(빈도 폐널티)이나 입력 프롬프트에 없는 단어를 생성

하는 것(존재 패널티)에 대해 모델에 패널티를 부과합니다. 이 값들을 조정함으로써 모델이 더 다양하고 관련성 있는 출력을 생성하도록 유도할 수 있습니다.

- **최대 길이:** 이 하이퍼파라미터는 모델이 단일 응답에서 생성할 수 있는 토큰(단어 또는 하위 단어)의 수에 대한 상한선을 설정합니다. 이는 생성된 텍스트의 장황함과 간결성을 제어하는 데 도움이 됩니다.

서로 다른 하이퍼파라미터 설정을 실험해보면서, 작은 조정만으로도 모델의 출력에 상당한 영향을 미칠 수 있다는 것을 알게 될 것입니다. 마치 요리법을 미세 조정하는 것과 같습니다 – 소금을 조금 더 넣거나 조리 시간을 약간 더 늘리는 것이 최종 요리의 맛을 크게 바꿀 수 있습니다.

핵심은 각 하이퍼파라미터가 모델의 동작에 어떤 영향을 미치는지 이해하고 특정 작업에 맞는 적절한 균형을 찾는 것입니다. 다양한 설정을 시도해보고 그것들이 생성된 텍스트에 어떤 영향을 미치는지 확인하는 것을 두려워하지 마세요. 시간이 지나면서, 어떤 하이퍼파라미터를 조정해야 하는지, 그리고 원하는 결과를 얻기 위해 어떻게 해야 하는지에 대한 직관이 생길 것입니다.

이러한 매개변수들을 프롬프트 엔지니어링, 검색 증강 생성, 그리고 미세 조정과 결합함으로써, 특정 사용 사례에 대해 더 정확하고, 관련성 있으며, 가치 있는 응답을 생성하도록 언어 모델의 경로를 좁히고 안내할 수 있습니다.

원본 모델과 명령어 튜닝 모델 비교

원본 모델은 LLM의 정제되지 않은, 훈련되지 않은 버전입니다. 이들을 특정 훈련의 영향을 받지 않은 깨끗한 캔버스라고 생각해보세요. 이들은 초기에 학습된 방대한 데이터를 기반으로 구축되어 있으며, 광범위한 출력을 생성할 수 있습니다. 하지만 명령어 기반의 미세 조정 레이어가 추가되지 않은 상태에서는, 그들의 응답은 예측하기 어려울 수 있으며 원하는 출력을 얻기 위해서는 더 세밀하고 신중하게 작성된 프롬프트가 필요합니다. 원본 모델과 작업하는 것은

마치 엄청난 양의 지식은 있지만, 매우 정확한 지시를 하지 않는 한 당신이 요구하는 것에 대한 직관이 전혀 없는 천재 바보와 소통하는 것과 같습니다. 그들은 종종 앵무새처럼 느껴지는데, 이는 이해할 만한 말을 하게 만들 수 있는 정도라 해도, 대부분 당신이 말한 것을 단순히 반복하는 경우가 많기 때문입니다.

반면에 명령어 튜닝 모델은 지시사항을 이해하고 따르도록 특별히 설계된 훈련 라운드를 거쳤습니다. GPT-4, Claude 3 및 다른 많은 인기 있는 LLM 모델들은 모두 강력한 명령어 튜닝이 되어 있습니다. 이 훈련은 모델에 원하는 결과와 함께 지시사항의 예시들을 제공하여, 모델이 다양한 명령을 해석하고 실행하는 방법을 효과적으로 가르칩니다. 결과적으로, 명령어 튜닝 모델은 프롬프트 뒤의 의도를 더 쉽게 이해하고 사용자의 기대에 더 잘 부합하는 응답을 생성할 수 있습니다. 이는 특히 광범위한 프롬프트 엔지니어링에 시간이나 전문성을 투자할 여유가 없는 사람들에게 더 사용자 친화적이고 다루기 쉽게 만듭니다.

원본 모델: 필터링되지 않은 캔버스

Llama 2-70B나 Yi-34B와 같은 원본 모델은 GPT-4와 같은 인기 있는 LLM을 사용하면서 익숙해졌을 수 있는 것보다 더 필터링되지 않은 모델 기능에 대한 접근을 제공합니다. 이러한 모델들은 특정 지시를 따르도록 사전 튜닝되어 있지 않아, 신중한 프롬프트 엔지니어링을 통해 모델의 출력을 직접 조작할 수 있는 빈 캔버스를 제공합니다. 이 접근 방식은 명시적인 지시 없이도 AI를 원하는 방향으로 안내할 수 있는 프롬프트를 만드는 방법에 대한 깊은 이해가 필요합니다. 이는 마치 모델의 응답을 해석하거나 안내하는 중간 레이어 없이 AI의 “원본” 레이어에 직접 접근하는 것과 같습니다(따라서 이름이 원본 모델입니다).

원본 모델의 과제는 반복적인 패턴에 빠지거나 무작위 출력을 생성하는 경향에 있습니다. 하지만 세심한 프롬프트 엔지니어링과 와 같은 매개변수 조정을 통해 원본 모델이 독특하고 창의적인 콘텐츠를 생성하도록 유도할 수 있습니다. 이

과정에는 trade-off가 따릅니다. 원본 모델이 혁신을 위한 타의 추종을 불허하는 유연성을 제공하지만, 더 높은 수준의 전문성을 요구합니다.

The screenshot shows a conversation interface. On the right, there is a blue message box containing a dialogue between Abbott and Costello. Abbott says, "You throw the ball to first base." Costello responds with "Then who gets it?" Abbott replies, "Naturally." Costello then says, "Naturally." Abbott follows up with "Now you've got it." Costello concludes with "I throw the ball to Naturally." Above the message box, the name "Obie" is visible next to a user icon. Below the message box, the text "GPT-4 Turbo Preview" is displayed, along with a small circular icon. The main content area contains a paragraph about the "Who's on First?" sketch, mentioning its rapid-fire verbal exchange and focus on confusing baseball player names. At the bottom left of the content area, the text "~42.7 tokens/s" is shown. On the far right, there are two small downward-pointing arrows.

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

그림 3. 비교를 위해, 동일한 모호한 프롬프트를 GPT-4에 입력한 결과

지시어 튜닝 모델: 안내된 경험

지시어 튜닝 과정은 인간이 생성한 지시 프롬프트와 응답의 대규모 말뭉치로 모델을 훈련시키는 것을 포함합니다. 주목할 만한 예시로 오픈소스 [databricks-dolly-15k](#) 데이터셋이 있는데, 이는 이 만든 15,000개 이상의 프롬프트/응답 쌍을 포함하고 있으며 직접 확인해볼 수 있습니다. 이 데이터셋은 , , , , 브레이스토밍을 포함한 8가지 다른 지시 카테고리를 다룹니다.

데이터 생성 과정에서 기여자들은 각 카테고리에 대한 프롬프트와 응답을 만드는 방법에 대한 가이드라인을 받았습니다. 예를 들어, 창의적 글쓰기 작업의 경우, 모델의 출력을 안내하기 위한 구체적인 제약 조건, 지시사항 또는 요구사항을 제공하도록 지시받았습니다. 폐쇄형 질문 답변의 경우, 주어진 위키피디

아 구절을 바탕으로 사실에 기반한 정확한 응답이 필요한 질문을 작성하도록 요청받았습니다.

결과적으로 만들어진 데이터셋은 대규모 언어 모델을 와 같은 시스템의 상호 작용적이고 지시를 따르는 기능을 갖추도록 미세 조정하는 데 귀중한 자원이 됩니다. 다양한 범위의 인간이 생성한 지시사항과 응답으로 훈련함으로써, 모델은 특정 지시사항을 이해하고 따르는 법을 배워 다양한 작업을 더 능숙하게 처리할 수 있게 됩니다.

직접적인 미세 조정 외에도, databricks-dolly-15k와 같은 데이터셋의 지시 프롬프트는 에도 사용될 수 있습니다. 기여자가 생성한 프롬프트를 대규모 오픈 언어 모델에 퓨샷 예제로 제출함으로써, 개발자들은 각 카테고리에서 훨씬 더 큰 규모의 지시사항 말뭉치를 생성할 수 있습니다. Self-Instruct 논문에서 설명된 이 접근 방식은 더 강력한 지시 따르기 모델을 만들 수 있게 합니다.

또한, 이러한 데이터셋의 지시문과 응답은 바꿔쓰기와 같은 기법을 통해 보강될 수 있습니다. 각 프롬프트나 짧은 응답을 다른 방식으로 진술하고 이를 해당하는 실제 샘플과 연관시킴으로써, 개발자들은 모델의 지시 수행 능력을 향상시키는 일종의 정규화를 도입할 수 있습니다.

명령어 조정 모델이 제공하는 사용 편의성은 일정 수준의 유연성을 희생하는 대가로 얻어집니다. 이러한 모델들은 대개 강력한 검열이 적용되어 있어, 특정 작업에 필요한 수준의 창의적 자유를 항상 제공하지는 못할 수 있습니다. 이들의 출력은 미세 조정 데이터에 내재된 편향과 한계에 크게 영향을 받습니다.

이러한 한계에도 불구하고, 명령어 조정 모델은 사용자 친화적인 특성과 최소한의 프롬프트 엔지니어링으로도 다양한 작업을 처리할 수 있는 능력 덕분에 점점 더 인기를 얻고 있습니다. 더 많은 고품질 지시 데이터셋이 사용 가능해짐에 따라, 이러한 모델들의 성능과 다용도성이 더욱 향상될 것으로 예상됩니다.

프로젝트에 적합한 모델 선택하기

기본 모델과 명령어 조정 모델 중 선택하는 것은 결국 프로젝트의 구체적인 요구사항에 달려 있습니다. 높은 수준의 창의성과 독창성이 요구되는 작업의

경우, 기본 모델은 혁신을 위한 강력한 도구가 됩니다. 이러한 모델들은 개발자들이 AI 기반 애플리케이션을 통해 달성할 수 있는 잠재력의 한계를 넓히면서 LLM의 잠재력을 충분히 탐구할 수 있게 해주지만, 더 많은 직접적인 관여와 실험 의지가 필요합니다. 온도와 같은 설정은 명령어 조정 모델보다 기본 모델에서 훨씬 더 큰 영향을 미칩니다.



프롬프트에 포함하는 모든 것이 기본 모델이 반복하려고 시도하는 것입니다. 예를 들어, 프롬프트가 채팅 기록인 경우, 기본 모델은 채팅을 계속 이어가려고 시도할 것입니다. max tokens 제한에 따라, 채팅의 다음 메시지만 생성하는 것이 아니라 자체적으로 전체 대화를 나눌 수도 있습니다!

Obie

Original: The movie was not very good.

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

Original: The food at the restaurant was okay.

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

Original: The weather today was kind of meh.

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

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

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

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

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

Original: This ice cream sucks!

Improved:

Mixtral 8x7B (base) ↗

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

...



그림 4. 문장 다시 쓰기의 Few-Shot 완성 예시를 보여주는 Mixtral 8x7B(기본)

Reddit 사용자 [phree_radical](#)이 제공한 위의 문장 다시 쓰기 예시를 준비하면서, 많은 실험을 거친 후에야 사용 가능한 결과를 얻을 수 있었으며, 최종적으로 다음과 같은 매개변수 설정으로 정착했습니다: 온도 0.08, Top P: 0.2, Top K: 1, 반복 패널티: 1.26.

프로덕션 환경에서 기본 모델로 이 접근 방식을 사용하려고 하면 `max_tokens` 매개변수의 강력한 영향력 때문에 까다로울 수 있습니다. 너무 짧게 설정하면 출력이 잘리고, 원하는 출력에 필요한 것보다 길게 설정하면 모델이 추가 예시들을 계속해서 환각할 것입니다.

결론적으로, 완전한 제어와 검열 부재가 정말로 필요한 경우가 아니라면, 명령어 조정 모델이 개발 프로세스를 상당히 간소화할 수 있습니다. 이 점을 더욱 명확히 하기 위해, 동일한 프롬프트에 대한 Mixtral 8x7B의 응답을 이번에는 명령어 조정 버전으로 살펴보겠습니다:

죄송하지만 이 아이스크림은 고품질 디저트에서 기대되는 풍부하고 크리미한 질감과 즐거운 맛이 부족하여 제 기대에 미치지 못합니다.
더 만족스럽고 즐거운 경험을 기대했었습니다.

주목할 만한 점은, `max tokens` 설정을 500으로 유지했음에도 모델이 추가적인 예시를 만들어내지 않고 원하는 출력의 끝에서 안정적으로 멈추었다는 것입니다.

프롬프트 엔지니어링

프로젝트에 AI를 적용하기 시작하면, 가장 중요하게 습득해야 할 기술 중 하나가 프롬프트 엔지니어링의 기술이라는 것을 곧 발견하게 될 것입니다. 하지만 프롬프트 엔지니어링이 정확히 무엇이고, 왜 그렇게 중요할까요?

핵심적으로, 프롬프트 엔지니어링은 언어 모델에 제공하는 입력 프롬프트를 설계하고 만드는 과정입니다. 이는 AI와 효과적으로 소통하는 방법을 이해하고, 지시사항, 예시, 맥락을 조합하여 모델이 원하는 응답을 생성하도록 유도하는 것입니다.

매우 지능적이지만 다소 문자 그대로 이해하는 친구와 대화하는 것처럼 생각해 보세요. 상호작용에서 최상의 결과를 얻으려면, 당신이 요청하는 것을 친구가

정확히 이해할 수 있도록 명확하고 구체적이며 충분한 맥락을 제공해야 합니다. 이것이 바로 프롬프트 엔지니어링이며, 처음에는 쉬워 보일 수 있지만, 믿어주세요, 숙달하기까지는 많은 연습이 필요합니다.

효과적인 프롬프트의 구성 요소

효과적인 프롬프트를 설계하기 위해서는 먼저 잘 만들어진 입력을 구성하는 주요 요소들을 이해해야 합니다. 다음은 필수적인 구성 요소들입니다:

1. **지시사항**: 모델에게 원하는 작업을 알려주는 명확하고 간단한 지시사항.
“다음 기사를 요약하세요”부터 “일몰에 대한 시를 생성하세요”, “이 프로젝트 변경 요청을 JSON 객체로 변환하세요” 등 다양할 수 있습니다.
2. **맥락**: 모델이 작업의 배경과 범위를 이해하는 데 도움이 되는 관련 정보.
여기에는 의도된 독자, 원하는 어조와 스타일, 또는 JSON 스키마와 같은 출력에 대한 특정 제약사항이나 요구사항이 포함될 수 있습니다.
3. **예시**: 원하는 출력의 유형을 보여주는 구체적인 예시들. 잘 선택된 몇 가지 예시를 제공함으로써, 모델이 원하는 응답의 패턴과 특성을 학습하는 데 도움을 줄 수 있습니다.
4. **입력 형식**: 줄바꿈과 마크다운 서식은 프롬프트에 구조를 부여합니다.
프롬프트를 단락으로 나누어 관련된 지시사항을 그룹화하면 인간과 AI 모두가 이해하기 쉬워집니다. 글머리 기호와 번호 매기기는 항목과 순서를 정의하게 해주며, 굵은 글씨와 이탤릭체는 강조를 표시하게 해줍니다.
5. **출력 형식**: 출력이 어떻게 구조화되고 형식화되어야 하는지에 대한 구체적인 지시사항. 여기에는 원하는 길이, 제목이나 글머리 기호의 사용, 마크다운 서식, 또는 따라야 할 기타 특정 출력 템플릿이나 규칙에 대한 지시사항이 포함될 수 있습니다.

이러한 구성 요소들을 다양한 방식으로 조합함으로써, 특정 요구사항에 맞춘 프롬프트를 만들고 모델이 고품질의 관련된 응답을 생성하도록 유도할 수 있습니다.

프롬프트 설계의 예술과 과학

효과적인 프롬프트를 만드는 것은 예술이자 과학입니다. (그래서 우리는 이것을 기술이라고 부릅니다.) 언어 모델의 능력과 한계에 대한 깊은 이해와 함께, 원하는 동작을 이끌어내는 프롬프트를 설계하는 창의적인 접근이 필요합니다. 적어도 저에게는 이러한 창의성이 이 일을 즐겁게 만듭니다. 특히 결정론적 동작을 추구할 때는 매우 좌절스러울 수도 있습니다.

프롬프트 엔지니어링의 한 가지 핵심 측면은 구체성과 유연성의 균형을 이해하는 것입니다. 한편으로는 모델을 올바른 방향으로 유도하기 위한 충분한 지침을 제공해야 합니다. 다른 한편으로는 옛지 케이스를 다룰 때 모델의 창의성과 유연성을 활용할 수 있는 능력을 제한하지 않도록 너무 규범적이지 않아야 합니다.

또 다른 중요한 고려사항은 예시의 사용입니다. 잘 선택된 예시는 모델이 원하는 출력의 유형을 이해하는 데 매우 강력한 도움이 될 수 있습니다. 하지만 예시를 신중하게 사용하고 원하는 응답을 대표할 수 있는지 확인하는 것이 중요합니다. 나쁜 예시는 최선의 경우에도 토큰의 낭비이며, 최악의 경우에는 원하는 출력을 망칠 수 있습니다.

프롬프트 엔지니어링 기법과 모범 사례

프롬프트 엔지니어링의 세계를 더 깊이 탐구하면서, 더 효과적인 프롬프트를 만드는 데 도움이 되는 다양한 기법과 모범 사례들을 발견하게 될 것입니다. 다음은 탐구해볼 만한 몇 가지 주요 영역입니다:

1. **제로샷 대 퓨샷 학습:** 제로샷 학습(예시를 제공하지 않음)과 원샷 또는 퓨샷 학습(소수의 예시를 제공)을 언제 사용할지 이해하는 것은 더 효율적이고 효과적인 프롬프트를 만드는 데 도움이 될 수 있습니다.
2. **반복적 개선:** 모델의 출력을 기반으로 프롬프트를 반복적으로 개선하는 과정은 최적의 프롬프트 설계에 도달하는 데 도움이 됩니다. [피드백](#)

루프는 언어 모델의 출력을 활용하여 생성된 콘텐츠의 품질과 관련성을 점진적으로 향상시키는 강력한 접근 방식입니다.

3. **프롬프트 체이닝**: 여러 프롬프트를 순차적으로 결합하면 복잡한 작업을 더 작고 관리하기 쉬운 단계로 분해할 수 있습니다. **프롬프트 체이닝**은 복잡한 작업이나 대화를 더 작고 상호 연결된 일련의 프롬프트로 분해하는 것을 포함합니다. 프롬프트들을 서로 연결함으로써, AI를 다단계 프로세스를 통해 안내하며 전체 상호작용에서 맥락과 일관성을 유지할 수 있습니다.
4. **프롬프트 튜닝**: 특정 도메인이나 작업에 맞게 프롬프트를 맞춤 설정하면 더 전문화되고 효과적인 프롬프트를 만들 수 있습니다. **프롬프트 템플릿**은 유연하고, 재사용 가능하며, 유지보수가 용이한 프롬프트 구조를 만들어 주어진 작업에 더 쉽게 적용할 수 있도록 도와줍니다.

제로샷, 원샷, 또는 퓨샷 러닝을 언제 사용할지 아는 것은 프롬프트 엔지니어링을 마스터하는 데 특히 중요한 부분입니다. 각 접근 방식에는 고유한 장단점이 있으며, 각각을 언제 사용해야 하는지 이해하면 더 효과적이고 효율적인 프롬프트를 만들 수 있습니다.

제로샷 러닝: 예시가 필요 없을 때

제로샷 러닝은 언어 모델이 어떠한 예시나 명시적인 훈련 없이도 작업을 수행할 수 있는 능력을 말합니다. 다시 말해, 모델에게 작업을 설명하는 프롬프트를 제공하면, 모델은 기존에 가지고 있는 지식과 언어에 대한 이해를 바탕으로 응답을 생성합니다.

제로샷 러닝은 다음과 같은 경우에 특히 유용합니다:

1. 작업이 비교적 단순하고 명확하며, 모델이 사전 훈련 과정에서 유사한 작업을 접했을 가능성이 높을 때
2. 추가적인 안내 없이 모델의 고유한 능력을 테스트하고 새로운 작업에 대한 반응을 보고 싶을 때

3. 다양한 작업과 도메인에 대해 훈련된 크고 다양한 언어 모델을 사용할 때

하지만 제로샷 러닝은 예측할 수 없을 수 있으며 항상 원하는 결과를 산출하지는 않을 수 있습니다. 모델의 응답은 사전 훈련 데이터의 편향이나 불일치의 영향을 받을 수 있으며, 더 복잡하거나 미묘한 작업에서는 어려움을 겪을 수 있습니다.

제가 본 제로샷 프롬프트 중에는 테스트 케이스의 80%에서는 잘 작동하지만 나머지 20%에서는 완전히 잘못되거나 이해할 수 없는 결과를 내는 경우가 있었습니다. 특히 제로샷 프롬프팅에 많이 의존하는 경우에는 철저한 테스트 체계를 구현하는 것이 매우 중요합니다.

원샷 러닝: 하나의 예시가 차이를 만들 때

원샷 러닝은 작업 설명과 함께 원하는 출력의 단일 예시를 모델에 제공하는 것을 포함합니다. 이 예시는 모델이 자체 응답을 생성하는 데 사용할 수 있는 템플릿이나 패턴으로 작용합니다.

원샷 러닝은 다음과 같은 경우에 효과적일 수 있습니다:

1. 작업이 비교적 새롭거나 특정적이어서 모델이 사전 훈련 중에 유사한 예시를 많이 접하지 않았을 때
2. 원하는 출력 형식이나 스타일을 명확하고 간결하게 보여주고 싶을 때
3. 작업이 작업 설명만으로는 명확하지 않을 수 있는 특정 구조나 규칙을 요구할 때



여러분에게 명확해 보이는 설명이 AI에게는 반드시 명확한 것은 아닐 수 있습니다. 원샷 예시는 이를 명확히 하는 데 도움이 될 수 있습니다.

원샷 러닝은 모델이 기대사항을 더 명확하게 이해하고 제공된 예시에 더 가깝게 부합하는 응답을 생성하는 데 도움이 될 수 있습니다. 하지만 예시를 신중하게 선택하고 그것이 원하는 출력을 대표하는지 확인하는 것이 중요합니다. 예시를 선택할 때는 잠재적인 엣지 케이스와 프롬프트가 처리할 입력의 범위에 대해 고려해야 합니다.

그림 5. 원하는 JSON의 원샷 예시

```
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 }
```

퓨샷 러닝: 여러 예시를 통한 성능 향상

퓨샷 러닝은 모델에 작업 설명과 함께 소수의 예시(일반적으로 2개에서 10개 사이)를 제공하는 것을 포함합니다. 이러한 예시들은 모델에 더 많은 맥락과

변형을 제공하여 더 다양하고 정확한 응답을 생성하는 데 도움을 줍니다.

퓨샷 러닝은 다음과 같은 경우에 특히 유용합니다:

1. 작업이 복잡하거나 미묘해서 단일 예시만으로는 모든 관련 측면을 포착하기에 충분하지 않을 때
2. 다양한 변형이나 예외적인 경우를 보여주는 여러 예시를 모델에 제공하고자 할 때
3. 작업이 특정 도메인이나 스타일에 부합하는 응답을 생성해야 할 때

여러 예시를 제공함으로써, 모델이 작업에 대해 더 견고한 이해를 발전시키고 더 일관되고 신뢰할 수 있는 응답을 생성하도록 도울 수 있습니다.

예시: 프롬프트는 상상 이상으로 복잡할 수 있습니다

오늘날의 대규모 언어 모델들은 여러분이 상상하는 것보다 훨씬 더 강력하고 추론 능력이 뛰어납니다. 따라서 프롬프트를 단순히 입력과 출력 쌍의 명세로만 생각하지 마세요. 사람과 상호작용하는 것처럼 길고 복잡한 지시사항을 실험해 볼 수 있습니다.

예를 들어, 제가 Olympia에서 아마도 세계에서 가장 큰 API 중 하나인 Google 서비스와의 통합을 프로토타이핑할 때 사용했던 프롬프트가 있습니다. 이전 실험들을 통해 GPT-4가 Google API에 대해 상당한 지식을 가지고 있다는 것이 입증되었고, 제가 AI에게 제공하고 싶은 각 기능을 일일이 구현하는 세분화된 매팅 레이어를 작성할 시간이나 동기가 없었습니다. AI에게 Google API 전체에 대한 접근 권한을 제공하면 어떨까요?

저는 AI에게 Google API 엔드포인트에 대한 직접적인 HTTP 접근 권한이 있으며, 사용자를 대신하여 Google 앱과 서비스를 사용하는 것이 그것의 역할이라고 알려주는 것으로 프롬프트를 시작했습니다. 그런 다음 가이드라인과 AI가 가장 어려워하는 것으로 보이는 **fields** 매개변수와 관련된 규칙, 그리고 몇 가지 API 관련 힌트(실제 퓨샷 프롬프팅)를 제공했습니다.

다음은 AI에게 제공된 `invoke_google_api` 함수의 사용법을 설명하는 전체 프롬프트입니다.

- 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
```

이 프롬프트가 실제로 작동하는지 궁금하실 수 있습니다. 간단한 답변은 '예'입니다. 인공지능이 처음부터 API를 완벽하게 호출하지는 못했습니다. 하지만 실수를 했을 때는 단순히 그 결과로 나온 오류 메시지를 호출 결과로 다시 피드백했습니다. 자신의 오류를 인지하게 되면, 인공지능은 실수에 대해 추론하고 다시 시도할 수 있었습니다. 대부분의 경우 두세 번 시도 만에 올바른 결과를 얻을 수 있었습니다.

물론, 이 프롬프트를 사용할 때 Google API가 반환하는 큰 JSON 구조는 매우 비효율적이므로, 실제 프로덕션 환경에서 이 접근 방식을 사용하는 것을 추천하지는 않습니다. 하지만 이 접근 방식이 작동했다는 사실 자체가 프롬프트 엔지니어링이 얼마나 강력할 수 있는지를 보여주는 증거라고 생각합니다.

실험과 반복

궁극적으로, 프롬프트를 어떻게 설계할지는 특정 작업, 원하는 출력의 복잡성, 그리고 사용하는 언어 모델의 능력에 따라 달라집니다.

프롬프트 엔지니어로서, 다양한 접근 방식을 실험하고 결과를 바탕으로 반복적으로 개선하는 것이 중요합니다. 제로샷 러닝으로 시작하여 모델의 성능을 확인해보세요. 출력이 일관적이지 않거나 만족스럽지 않다면, 하나 이상의 예시를 제공하고 성능이 개선되는지 확인해보세요.

각각의 접근 방식 내에서도 변형과 최적화의 여지가 있다는 점을 기억하세요. 다양한 예시를 시도해보거나, 작업 설명의 표현을 조정하거나, 모델의 응답을 더 잘 유도하기 위한 추가 컨텍스트를 제공할 수 있습니다.

시간이 지나면서, 주어진 작업에 어떤 접근 방식이 가장 효과적일지에 대한 직관이 발달할 것이며, 더 효과적이고 효율적인 프롬프트를 만들 수 있게 될 것입니다. 핵심은 프롬프트 엔지니어링에 대해 호기심을 가지고, 실험적이며, 반복적인 접근 방식을 유지하는 것입니다.

이 책을 통해, 우리는 이러한 기술들을 더 깊이 살펴보고 실제 시나리오에서 어떻게 적용할 수 있는지 알아볼 것입니다. 프롬프트 엔지니어링의 예술과 과학을 마스터함으로써, AI 기반 애플리케이션 개발의 잠재력을 최대한 활용할 수 있게 될 것입니다.

모호함의 예술

대규모 언어 모델(LLM)을 위한 효과적인 프롬프트를 만들 때, 더 구체적이고 상세한 지침이 더 나은 결과를 가져온다고 흔히 가정합니다. 하지만 실제 경험에 따르면 이것이 항상 사실은 아닙니다. 실제로, 프롬프트에서 의도적으로 모호한 표현을 사용하는 것이 종종 더 나은 결과를 가져올 수 있으며, 이는 LLM의 일반화와 추론 능력을 활용하는 것입니다.

5억 개 이상의 GPT 토큰을 처리한 경험이 있는 스타트업 창업자 Ken은 [자신의 경험에서 얻은 귀중한 통찰](#)을 공유했습니다. 그가 배운 핵심 교훈 중 하나는 프롬프트에 있어서 “적을수록 좋다”는 것이었습니다. 정확한 목록이나 지나치게 상세한 지침 대신, Ken은 LLM이 기본 지식에 의존하도록 하는 것이 종종 더 나은 결과를 산출한다는 것을 발견했습니다.

이러한 깨달음은 모든 것을 세밀하게 명시해야 하는 전통적인 코딩 사고방식을 뒤집는 것입니다. LLM의 경우, 그들이 방대한 양의 지식을 보유하고 있으며 지능적인 연결과 추론을 할 수 있다는 점을 인식하는 것이 중요합니다. 프롬프트에서 더 모호한 표현을 사용함으로써, LLM이 자신의 이해력을 활용하여 명시적으로 지정하지 않은 해결책을 도출할 수 있는 자유를 제공하게 됩니다.

예를 들어, Ken의 팀이 텍스트를 50개 미국 주 또는 연방 정부 중 하나에 관련된 것으로 분류하는 파이프라인을 작업할 때, 그들의 초기 접근 방식은 JSON 형식의 배열로 된 주들의 전체 상세 목록과 해당 ID를 제공하는 것이었습니다.

- 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} ...]

접근 방식이 실패를 거듭하여 프롬프트를 개선하기 위해 더 깊이 파고들어야 했습니다. 그 과정에서 LLM이 ID를 종종 잘못 인식했음에도 불구하고, 명시적으로 요청하지 않았는데도 name 필드에 올바른 주의 전체 이름을 일관되게 반환한다는 것을 발견했습니다.

지역 ID를 제거하고 “GPT, 당신이 50개 주를 알고 있다는 것은 분명하니, 이것이 해당되는 주의 전체 이름만 알려주거나, 미국 정부에 관한 것이라면 Federal이라고 말해주세요”와 같이 프롬프트를 단순화함으로써 더 나은 결과를 얻었습니다. 이러한 경험은 LLM의 일반화 능력을 활용하고 기존 지식을 바탕으로 추론하도록 허용하는 것의 힘을 잘 보여줍니다.

Ken이 이러한 분류 접근 방식을 더 전통적인 프로그래밍 기법 대신 선택한 이유는 LLM 기술의 잠재력을 받아들인 우리의 사고방식을 잘 보여줍니다: “이것은 어려운 작업이 아닙니다 – 문자열/정규식을 사용할 수도 있었지만, 이상한 예외 케이스가 너무 많아서 더 오래 걸렸을 것입니다.”

LLM이 더 모호한 프롬프트를 받았을 때 품질과 일반화 능력이 향상되는 것은 고차원적 사고와 위임의 주목할 만한 특징입니다. 이는 LLM이 모호성을 처리하고 주어진 맥락을 바탕으로 지능적인 결정을 내릴 수 있다는 것을 보여줍니다. 하지만 모호하다는 것이 불명확하거나 애매하다는 의미는 아닙니다. 핵심은

LLM이 자신의 지식과 일반화 능력을 활용할 수 있는 유연성을 제공하면서도 올바른 방향으로 이끌 수 있는 충분한 맥락과 안내를 제공하는 것입니다.

따라서 프롬프트를 설계할 때는 다음과 같은 “적을수록 좋다”는 팁을 고려하세요:

1. 과정의 모든 세부사항을 지정하기보다는 원하는 결과에 집중하세요.
2. 관련된 맥락과 제약사항을 제공하되, 과도한 세부사항은 피하세요.
3. 일반적인 개념이나 엔티티를 참조하여 기존 지식을 활용하세요.
4. 주어진 맥락을 바탕으로 추론과 연결이 가능하도록 여지를 남기세요.
5. LLM의 응답을 바탕으로 프롬프트를 반복하고 개선하여 구체성과 모호성 사이의 적절한 균형을 찾으세요.

프롬프트 엔지니어링에서 모호성의 예술을 받아들임으로써 LLM의 잠재력을 최대한 끌어낼 수 있습니다. LLM의 일반화 능력과 지능적 결정을 신뢰하면 받게 되는 출력의 품질과 창의성에 놀랄 수 있습니다. 다양한 모델이 프롬프트의 구체성 수준에 따라 어떻게 반응하는지 주의 깊게 관찰하고 그에 맞게 조정하세요. 연습과 경험을 통해 언제 더 모호하게 하고 언제 추가 지침을 제공해야 하는지에 대한 예리한 감각을 개발할 수 있으며, 이를 통해 애플리케이션에서 LLM의 힘을 효과적으로 활용할 수 있습니다.

프롬프트 엔지니어링에서 인격화가 지배적인 이유

인격화, 즉 비인간 개체에 인간의 특성을 부여하는 것은 대규모 언어 모델을 위한 프롬프트 엔지니어링에서 의도적인 이유로 지배적인 접근 방식입니다. 이는 강력한 AI 시스템과의 상호작용을 더 직관적이고 광범위한 사용자(우리 애플리케이션 개발자 포함)가 접근하기 쉽게 만드는 디자인 선택입니다.

LLM을 인격화하면 시스템의 기술적 복잡성을 전혀 모르는 사람들도 즉시 이해할 수 있는 프레임워크를 제공합니다. 명령어 조정이 되지 않은 모델로 유용한 작업을 하려고 시도해보면 알 수 있듯이, 가치 있는 결과를 제공하는

연속성을 기대할 수 있는 프레임을 구성하는 것은 어려운 작업입니다. 이는 소수의 전문가만이 보유하고 있는 시스템의 내부 작동 방식에 대한 깊은 이해가 필요합니다.

언어 모델과의 상호작용을 두 사람 간의 대화로 취급함으로써, 우리는 우리의 필요와 기대를 전달하기 위해 인간 의사소통에 대한 우리의 타고난 이해를 활용할 수 있습니다. 초기 매킨토시 UI 디자인이 정교함보다 즉각적인 직관성을 우선시했던 것처럼, AI의 인격화 프레임은 자연스럽고 친숙한 방식으로 참여할 수 있게 해줍니다.

다른 사람과 의사소통할 때, 우리는 본능적으로 “당신”이라는 표현을 사용하여 직접 말을 걸고 그들이 어떻게 행동하기를 기대하는지에 대한 명확한 방향을 제시합니다. 이는 시스템 프롬프트를 지정하고 대화식 상호작용을 하면서 AI의 행동을 안내하는 프롬프트 엔지니어링 과정으로 자연스럽게 이어집니다.

이러한 방식으로 상호작용을 프레임화함으로써, AI에 지침을 제공하고 그에 대한 관련된 응답을 받는 개념을 쉽게 이해할 수 있습니다. 인격화 접근 방식은 인지적 부담을 줄이고 시스템의 기술적 복잡성과 싸울 때 대신 당면한 작업에 집중할 수 있게 해줍니다.

인격화가 AI 시스템을 더 접근하기 쉽게 만드는 강력한 도구이지만, 특정한 위험과 한계도 있다는 점을 주목해야 합니다. 사용자가 비현실적인 기대를 하거나 우리 시스템에 불건전한 감정적 애착을 형성할 수 있습니다. 프롬프트 엔지니어와 개발자로서, 인격화의 이점을 활용하면서도 사용자가 AI의 능력과 한계를 명확히 이해할 수 있도록 균형을 맞추는 것이 중요합니다.

프롬프트 엔지니어링 분야가 계속 발전함에 따라, 대규모 언어 모델과의 상호 작용 방식에서 더 많은 개선과 혁신을 기대할 수 있습니다. 그러나 직관적이고 접근하기 쉬운 개발자 및 사용자 경험을 제공하기 위한 수단으로서의 의인화는 아마도 이러한 시스템 설계의 기본 원칙으로 남을 것입니다.

지침과 데이터의 분리: 핵심 원칙

이러한 시스템의 보안과 신뢰성을 뒷받침하는 근본적인 원칙을 이해하는 것이 필수적입니다: 바로 지침과 데이터의 분리입니다.

전통적인 컴퓨터 과학에서, 수동적 데이터와 능동적 지침 간의 명확한 구분은 핵심적인 보안 원칙입니다. 이러한 분리는 시스템의 무결성과 안정성을 손상시킬 수 있는 의도치 않은 또는 악의적인 코드 실행을 방지하는 데 도움이 됩니다. 하지만 오늘날의 LLM들은 주로 챗봇과 같은 지침 수행 모델로 개발되어 왔기 때문에, 종종 이러한 공식적이고 원칙적인 분리가 부족합니다.

LLM의 관점에서 보면, 지침은 시스템 프롬프트든 사용자가 제공한 프롬프트든 입력의 어디에나 나타날 수 있습니다. 이러한 분리의 부재는 SQL 인젝션이나 적절한 메모리 보호가 없는 운영 체제가 직면한 문제와 유사한 잠재적 취약점과 바람직하지 않은 동작을 초래할 수 있습니다.

LLM을 사용할 때는 이러한 한계를 인식하고 위험을 완화하기 위한 조치를 취하는 것이 중요합니다. 한 가지 접근 방식은 지침과 데이터를 명확히 구분하도록 프롬프트와 입력을 신중하게 작성하는 것입니다. 무엇이 지침이고 무엇이 수동적 데이터로 취급되어야 하는지에 대한 명시적 안내를 제공하는 일반적인 방법에는 마크업 스타일 태깅이 포함됩니다. 프롬프트를 통해 LLM이 이러한 분리를 더 잘 이해하고 존중하도록 도울 수 있습니다.

그림 6. XML을 사용하여 지침, 소스 자료, 사용자의 프롬프트를 구분하기

```
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>
```

또 다른 기법은 LLM에 제공되는 입력값에 대한 추가적인 검증 및 살균 계층을 구현하는 것입니다. 데이터에 포함될 수 있는 잠재적인 지시사항이나 코드 스니펫을 필터링하거나 이스케이프 처리함으로써 의도하지 않은 실행의 가능성을 줄일 수 있습니다. [프롬프트 체이닝](#)과 같은 패턴이 이러한 목적으로 유용합니다.

더불어, 애플리케이션 아키텍처를 설계할 때는 더 높은 수준에서 지시사항과 데이터의 분리를 강제하는 메커니즘을 고려해야 합니다. 여기에는 지시사항과 데이터를 처리하기 위한 별도의 엔드포인트나 API를 사용하고, 엄격한 입력 검증과 구문 분석을 구현하며, 최소 권한의 원칙을 적용하여 LLM이 접근하고 실행할 수 있는 범위를 제한하는 것이 포함됩니다.

최소 권한의 원칙

최소 권한의 원칙을 적용하는 것은 손님들이 꼭 필요한 방에만 접근할 수 있는 매우 독특한 파티를 여는 것과 같습니다. 거대한 저택에서 파티를 열고 있다고 상상해보세요. 모든 사람이 와인 저장고나 주인 침실에 들어갈 필요는 없겠죠? 이 원칙을 적용함으로써, 특정 문만 열 수 있는 열쇠를 나눠주는 것과 같이, 각 손님 또는 우리의 경우 LLM 애플리케이션의 각 구성 요소가 자신의 역할을 수행하는 데 필요한 접근 권한만을 가지도록 보장합니다.

이는 단순히 열쇠를 아끼는 것이 아니라, 위협이 어디서든 올 수 있는 세상에서 활동 영역을 제한하는 것이 현명한 선택임을 인정하는 것입니다. 초대받지 않은 사람이 파티에 들어온다 하더라도, 비유하자면 현관에만 갇혀 있게 되어 일으킬 수 있는 문제가 크게 제한됩니다. 따라서 LLM 애플리케이션을 보호할 때는 기억하세요: 필요한 방의 열쇠만 제공하고 저택의 나머지 부분은 안전하게 지키세요. 이는 단순한 예의가 아닌 훌륭한 보안 방식입니다.

현재 LLM의 상태가 지시사항과 데이터의 공식적인 분리를 가지고 있지 않을 수 있지만, 개발자로서 이러한 한계를 인식하고 위험을 완화하기 위한 선제적 조치를 취하는 것이 필수적입니다. 전통적인 컴퓨터 과학의 모범 사례를 적용하고 이를 LLM의 고유한 특성에 맞게 조정함으로써, 이러한 모델의 힘을 활용하면서도 시스템의 무결성을 유지하는 더욱 안전하고 신뢰할 수 있는 애플리케이션을 구축할 수 있습니다.

프롬프트 종류

완벽한 프롬프트를 작성하는 것은 종종 대상 도메인과 언어 모델의 뉘앙스에 대한 깊은 이해가 필요한 도전적이고 시간이 많이 소요되는 작업입니다. 이때 “프롬프트 종류” 기법이 등장하며, 대규모 언어 모델(LLM)의 기능을 활용하여

프로세스를 간소화하고 최적화하는 강력한 프롬프트 엔지니어링 접근 방식을 제공합니다.

프롬프트 종류는 LLM을 사용하여 프롬프트의 생성, 개선 및 최적화를 지원하는 다단계 기법입니다. 이 접근 방식은 인간의 전문성과 직관에만 의존하는 대신 LLM의 지식과 생성 능력을 활용하여 협력적으로 고품질 프롬프트를 만듭니다.

생성, 개선 및 통합의 반복적인 프로세스를 통해 프롬프트 종류를 사용하면 원하는 작업이나 출력과 더 잘 부합하는 일관성 있고 포괄적인 프롬프트를 만들 수 있습니다. 종류 프로세스는 OpenAI나 Anthropic과 같은 주요 AI 업체들이 제공하는 여러 “플레이그라운드” 중 하나에서 수동으로 수행하거나, 사용 사례에 따라 애플리케이션 코드의 일부로 자동화할 수 있습니다.

작동 방식

프롬프트 종류는 일반적으로 다음 단계들을 포함합니다:

1. **핵심 의도 파악:** 프롬프트를 분석하여 주요 목적과 원하는 결과를 결정합니다. 불필요한 정보를 제거하고 프롬프트의 핵심 의도에 집중합니다.
2. **모호성 제거:** 모호하거나 불분명한 언어가 있는지 프롬프트를 검토합니다. 의미를 명확히 하고 AI가 정확하고 관련성 있는 응답을 생성하도록 구체적인 세부 사항을 제공합니다.
3. **언어 단순화:** 명확하고 간결한 언어를 사용하여 프롬프트를 단순화합니다. AI를 혼란스럽게 하거나 노이즈를 발생시킬 수 있는 복잡한 문장 구조, 전문 용어 또는 불필요한 세부 사항을 피합니다.
4. **관련 컨텍스트 제공:** AI가 프롬프트를 효과적으로 이해하고 처리하는 데 필요한 가장 관련성 있는 컨텍스트 정보만 포함합니다. 핵심 의도에서 벗어날 수 있는 관련 없거나 중복된 세부 사항은 포함하지 않습니다.
5. **반복 및 개선:** AI의 응답과 피드백을 바탕으로 프롬프트를 지속적으로 반복하고 개선합니다. 생성된 출력을 평가하고 프롬프트의 명확성과 효과성을 향상시키기 위해 필요한 조정을 합니다. 선택적으로 [프롬프트 객](#)

체를 사용하여 데이터베이스에서 프롬프트 버전을 관리함으로써 반복을 추적하고 런타임에서 쉽게 변경 사항을 롤백할 수 있습니다.

초기 프롬프트 생성

프롬프트 디스틸레이션의 첫 단계는 원하는 작업이나 출력에 대한 상위 수준의 설명이나 의도를 LLM에 제공하는 것입니다. 이는 프롬프트가 달성해야 할 본질을 담은 간단한 설명이나 키워드의 집합이 될 수 있습니다. 그러면 LLM은 이 입력을 바탕으로 적절하다고 판단되는 관련 지침, 예시 또는 맥락을 포함하여 초기 프롬프트를 생성합니다.

Claude 3 Opus에 다음과 같은 초기 프롬프트를 제공해 보겠습니다:

- 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.

클로드가 다음과 같이 응답합니다:

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.

프롬프트 정제

초기 프롬프트가 생성되면, 이를 검토한 후 LLM에게 추가적인 맥락, 제약 조건 또는 피드백을 고려하도록 요청합니다. LLM이 제공된 입력과 작업에 대한 자체적인 이해를 바탕으로 정제된 버전의 프롬프트를 생성하도록 합니다. 이러한 정제 과정은 반복적으로 수행될 수 있으며, 각 반복마다 사용자와 LLM 간의 협력을 통해 프롬프트를 더욱 개선해 나갈 수 있습니다.

다음은 이전의 종류 과정을 이어가는 제 정제 요청입니다.

- 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.

그리고 Claude의 응답:

- 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.

프롬프트 압축

여러분도 이미 눈치채셨겠지만, LLM은 간단히 말하라고 요청하지 않는 한 꽤
장황한 편입니다. 이전 섹션에서 다뤘던 프롬프트가 상당히 길었으니, Claude를
통해 이를 줄여보도록 하겠습니다:

- 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.

Claude가 다음과 같이 훨씬 더 짧은 프롬프트로 응답합니다:

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

시스템 지시어와 컨텍스트 통합

프롬프트 자체를 개선하는 것 외에도, LLM은 최종 출력을 안내하기 위한 적절한 시스템 지시어나 컨텍스트 정보를 생성할 수 있습니다. 애플리케이션 코드에 통합될 AI 루틴을 프롬프트 엔지니어링할 때, 이 정제 단계에서는 거의 확실히 출력 제약에 초점을 맞추게 될 것입니다. 하지만 원하는 톤, 스타일, 형식 또는 생성된 응답에 영향을 미치는 다른 관련 매개변수들도 작업할 수 있습니다.

최종 프롬프트 조립

프롬프트 정제 과정의 정점은 최종 프롬프트의 조립입니다. 이는 개선된 프롬프트, 생성된 시스템 지시어, 그리고 통합된 컨텍스트를 원하는 출력을 생성하기 위해 사용할 준비가 된 일관성 있고 포괄적인 코드로 결합하는 것을 포함합니다.



최종 프롬프트 조립 단계에서 다시 한 번 프롬프트 압축을 실험해볼 수 있습니다. LLM에게 프롬프트의 본질적인 동작은 유지하면서 가능한 한 가장 짧은 토큰 시리즈로 문구를 줄이도록 요청하면 됩니다. 확실히 성공 여부는 불확실하지만, 특히 대규모로 실행될 프롬프트의 경우 효율성 향상으로 토큰 소비에서 상당한 비용을 절약할 수 있습니다.

주요 이점

LLM의 지식과 생성 능력을 활용하여 프롬프트를 개선함으로써, 결과적으로 나오는 프롬프트는 더 잘 구조화되고, 정보가 풍부하며, 특정 작업에 맞춤화될 가능성이 높습니다. 반복적인 개선 과정은 프롬프트의 품질을 보장하고 원하는 의도를 효과적으로 포착하는 데 도움이 됩니다. 다른 이점들은 다음과 같습니다:

효율성과 속도: 프롬프트 정제는 프롬프트 생성과 개선의 특정 측면을 자동화 함으로써 프롬프트 엔지니어링 과정을 간소화합니다. 이 기법의 협력적 특성은 효과적인 프롬프트로 더 빠르게 수렴하도록 하여, 수동 프롬프트 작성에 필요한 시간과 노력을 줄여줍니다.

일관성과 확장성: 프롬프트 엔지니어링 과정에서 LLM을 사용하면 이전의 성공적인 프롬프트로부터 모범 사례와 패턴을 학습하고 적용할 수 있어 프롬프트 전반에 걸쳐 일관성을 유지할 수 있습니다. 이러한 일관성과 대규모 프롬프트 생성 능력이 결합되어 프롬프트 정제는 대규모 AI 기반 애플리케이션에 유용한 기법이 됩니다.



프로젝트 아이디어: 애플리케이션 코드의 일부로 자동화된 프롬프트 정제를 수행하는 시스템에서 프롬프트 버전 관리와 등급 매기기 과정을 단순화하는 라이브러리 수준의 도구 개발.

프롬프트 정제를 구현하기 위해, 개발자들은 프롬프트 엔지니어링 과정의 다양한 단계에서 LLM을 통합하는 워크플로우나 파이프라인을 설계할 수 있습니다. 이는 API 호출, 맞춤형 도구, 또는 프롬프트 생성 중 사용자와 LLM 간의 원활한 상호작용을 촉진하는 통합 개발 환경을 통해 달성할 수 있습니다. 구체적인 구현 세부사항은 선택한 LLM 플랫폼과 애플리케이션의 요구사항에 따라 달라질 수 있습니다.

파인튜닝은 어떤가요?

이 책에서는 프롬프트 엔지니어링과 RAG를 광범위하게 다루지만, 파인튜닝은 다루지 않습니다. 이러한 결정의 주된 이유는, 제 의견으로는, 대부분의 애플리케이션 개발자들이 AI 통합 요구사항을 위해 파인튜닝이 필요하지 않기 때문입니다.

제로샷이나 퓨샷 예제, 제약 조건, 그리고 지시사항을 포함하여 신중하게 프롬프트를 작성하는 프롬프트 엔지니어링은 다양한 작업에 대해 관련성 있고 정확한 응답을 생성하도록 모델을 효과적으로 안내할 수 있습니다. 명확한 컨텍스트를 제공하고 잘 설계된 프롬프트를 통해 경로를 좁힘으로써, 파인튜닝 없이도 대규모 언어 모델의 방대한 지식을 활용할 수 있습니다.

마찬가지로, 검색 증강 생성(RAG)은 애플리케이션에 AI를 통합하는 강력한 접근 방식을 제공합니다. 외부 지식 베이스나 문서에서 관련 정보를 동적으로 검색함으로써, RAG는 프롬프트 시점에 모델에 집중된 컨텍스트를 제공합니다. 이를 통해 모델은 시간과 리소스가 많이 소요되는 파인튜닝 과정 없이도 더 정확하고, 최신이며, 도메인에 특화된 응답을 생성할 수 있습니다.

파인튜닝은 높은 수준의 맞춤화가 필요한 매우 전문화된 도메인이나 작업에 유용할 수 있지만, 흔히 상당한 계산 비용, 데이터 요구사항, 그리고 유지보수 부담이 따릅니다. 대부분의 애플리케이션 개발 시나리오에서는 효과적인 프롬프트 엔지니어링과 RAG의 조합만으로도 원하는 AI 기반 기능과 사용자 경험을 달성하기에 충분할 것입니다.

검색 증강 생성(RAG)

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

검색 증강 생성이란 무엇인가?

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

RAG는 어떻게 작동하는가?

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

왜 애플리케이션에서 RAG를 사용해야 하는가?

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

애플리케이션에 RAG 구현하기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

지식 소스 준비 (청킹)

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

명제 청킹

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

구현 참고사항

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

품질 검사

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

명제 기반 검색의 이점

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

RAG의 실제 사례

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

사례 연구: 임베딩 없는 세금 신고 애플리케이션의 RAG

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

지능형 쿼리 최적화 (IQO)

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

재순위화

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

RAG 평가 (RAGAs)

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

충실도

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

답변 관련성

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

컨텍스트 정확도

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

컨텍스트 연관성

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

컨텍스트 재현율

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

컨텍스트 개체 재현율

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

답변 의미 유사도 (ANSS)

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

답변 정확도

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

측면 비평

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

도전 과제와 미래 전망

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

의미적 청킹: 문맥 인식 분할로 검색 향상

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

계층적 인덱싱: 향상된 검색을 위한 데이터 구조화

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

자가 RAG: 자기 성찰적 개선

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

HyDE: 가상 문서 임베딩

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

대조 학습이란 무엇인가?

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

다수의 작업자



나는 AI 컴포넌트를 특정 작업을 수행하거나 복잡한 결정을 내리기 위해 애플리케이션 로직에 원활하게 통합될 수 있는, 거의 인간에 가까운 가상의 “작업자”로 생각하기를 좋아합니다. 이는 LLM의 능력을 의도적으로 인간화하여, 누구도 그들이 실제로 가지고 있지 않은 마법 같은 특성을 부여하는 것에 지나치게 흥분하지 않도록 하기 위함입니다.

복잡한 알고리즘이나 시간이 많이 소요되는 수동 구현에만 의존하는 대신, 개발자들은 AI 컴포넌트를 지능적이고 협신적이며 인간과 같은 존재로 개념화할 수 있습니다. 이들은 필요할 때마다 호출되어 복잡한 문제를 해결하고 자신들의 훈련과 지식을 바탕으로 해결책을 제공할 수 있습니다. 이러한 존재들은 주의가 산만해지거나 병가를 내지 않습니다. 그들은 지시받은 방식과 다르게 일을 처리하기로 갑자기 결정하지 않으며, 일반적으로 올바르게 프로그래밍되었다면 실수도 하지 않습니다.

기술적인 관점에서, 이 접근 방식의 핵심 원칙은 복잡한 작업이나 의사결정 과정을 전문 AI 작업자가 처리할 수 있는 더 작고 관리하기 쉬운 단위로 분해하는 것입니다. 각 작업자는 문제의 특정 측면에 집중하도록 설계되어 있으며, 자신만의 고유한 전문성과 능력을 제공합니다. 여러 AI 작업자들에게 작업량을 분산함으로써, 애플리케이션은 더 큰 효율성, 확장성, 그리고 적응성을 달성할 수 있습니다.

예를 들어, 사용자 생성 콘텐츠의 실시간 모더레이션이 필요한 웹 애플리케이션을 생각해보세요. 포괄적인 모더레이션 시스템을 처음부터 구현하는 것은 상당한 개발 노력과 지속적인 유지보수가 필요한 어려운 작업일 것입니다. 하지만 다수의 작업자 접근 방식을 사용하면, 개발자들은 AI 기반 모더레이션 작업자들을 애플리케이션 로직에 통합할 수 있습니다. 이러한 작업자들은 자동으로 부적절한 콘텐츠를 분석하고 플래그를 지정할 수 있어, 개발자들이 애플리케이션의 다른 중요한 측면에 집중할 수 있게 해줍니다.

독립적이고 재사용 가능한 컴포넌트로서의 AI 작업자

다수의 작업자 접근 방식의 핵심 측면은 모듈성입니다. 객체 지향 프로그래밍의 지지자들은 수십 년 동안 객체 간의 상호작용을 메시지로 생각하라고 말해왔습니다. AI 작업자들은 마치 실제 작은 인간들이 서로 대화하는 것처럼 일반 언어 메시지를 통해 “서로 대화할 수 있는” 독립적이고 재사용 가능한 컴포넌트로 설계될 수 있습니다. 이러한 느슨하게 결합된 접근 방식을 통해 새로운 AI 기술이 등장하거나 비즈니스 로직 요구사항이 변경될 때 애플리케이션이 적응하고 발전할 수 있습니다.

실제로, AI 작업자가 관련되어 있다고 해서 컴포넌트 간의 명확한 인터페이스와 통신 프로토콜을 설계해야 하는 필요성이 변한 것은 아닙니다. 여전히 성능, 확장성, 보안과 같은 다른 요소들도 고려해야 하지만, 이제는 완전히 새로운 “소프트 요구사항”도 고려해야 합니다. 예를 들어, 많은 사용자들은 자신들의 개인 데이터가 새로운 AI 모델 훈련에 사용되는 것을 반대합니다. 사용 중인 모델 제공업체가 제공하는 개인정보 보호 수준을 확인해보셨나요?

マイクロサービス로서의 AI 작업자?

다수의 작업자 접근 방식에 대해 읽다 보면, 마이크로서비스 아키텍처와 몇 가지 유사점이 있다는 것을 알 수 있습니다. 둘 다 복잡한 시스템을 더 작고, 더 관리하기 쉬우며, 독립적으로 배포 가능한 단위로 분해하는 것을 강조합니다. 마이크로서비스가 느슨하게 결합되고, 특정 비즈니스 기능에 중점을 두며, 잘 정의된 API를 통해 통신하도록 설계된 것처럼, AI 작업자들도 모듈화되고, 특정 작업에 특화되어 있으며, 명확한 인터페이스와 통신 프로토콜을 통해 서로 상호작용하도록 설계되어 있습니다.

하지만 명심해야 할 몇 가지 주요 차이점이 있습니다. 마이크로서비스는 일반적으로 서로 다른 기계나 컨테이너에서 실행되는 별도의 프로세스나 서비스로 구현되는 반면, AI 작업자는 특정 요구사항과 확장성 필요에 따라 단일 애플리케이션 내의 독립 컴포넌트로 구현되거나 별도의 서비스로 구현될 수 있습니다. 또한, AI 작업자 간의 통신은 마이크로서비스에서 일반적으로 사용되는 더 구조화된 데이터 형식이 아닌, 프롬프트, 지시사항, 생성된 콘텐츠와 같은 풍부한 자연어 기반 정보를 교환하는 것을 포함합니다.

이러한 차이점에도 불구하고, 모듈성, 느슨한 결합, 명확한 통신 인터페이스의 원칙은 두 패턴 모두에서 중심적인 역할을 합니다. 이러한 원칙들을 AI 작업자 아키텍처에 적용함으로써, 복잡한 문제를 해결하고 사용자에게 가치를 전달하기 위해 AI의 힘을 활용하는 유연하고, 확장 가능하며, 유지보수가 용이한 시스템을 만들 수 있습니다.

다수의 작업자 접근 방식은 다양한 도메인과 애플리케이션에 적용될 수 있으며, AI의 힘을 활용하여 복잡한 작업을 처리하고 지능적인 솔루션을 제공할 수 있습니다. AI 작업자들이 다양한 상황에서 어떻게 활용될 수 있는지 몇 가지 구체적인 예를 살펴보겠습니다.

계정 관리

실질적으로 모든 독립형 웹 애플리케이션에는 계정(또는 사용자)이라는 개념이 있습니다. Olympia에서는 사용자 계정과 관련된 다양한 종류의 변경 요청을 처리할 수 있도록 프로그래밍된 AccountManager AI 작업자를 사용합니다.

이 지시문은 다음과 같습니다:

- 1 You are an intelligent account manager **for Olympia**. The user will request changes to their account, **and** you will process those changes by invoking 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 results, plus guidance about how to proceed (**if** applicable). **If** you have 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.

`account.to_directive`로 생성된 계정의 초기 상태는 사용자, 구독 등과 같은 관련 데이터를 포함한 계정에 대한 단순한 텍스트 설명입니다.

AccountManager가 사용할 수 있는 다양한 기능을 통해 사용자의 구독을 수정하고, AI 컨설턴트와 기타 유료 부가 기능을 추가 및 제거하며, 계정 소유자에게 알림 이메일을 보낼 수 있습니다. **finished** 기능 외에도, 처리 중 오류가 발생하거나 요청에 대한 기타 도움이 필요한 경우 `notify_human_administrator`를 통해 관리자에게 알릴 수 있습니다.

질문이 있을 경우, AccountManager는 Olympia의 지식 베이스를 검색할 수 있으며, 여기서 예외적인 상황이나 진행 방법이 불확실한 기타 상황을 처리하는 방법에 대한 지침을 찾을 수 있습니다.

이커머스 애플리케이션

이커머스 영역에서 AI 작업자는 사용자 경험을 향상시키고 비즈니스 운영을 최적화하는 데 중요한 역할을 할 수 있습니다. 다음은 AI 작업자를 활용할 수 있는 몇 가지 방법입니다:

상품 추천

이커머스에서 AI 작업자의 가장 강력한 응용 분야 중 하나는 개인화된 상품 추천을 생성하는 것입니다. 사용자 행동, 구매 기록, 선호도를 분석함으로써, 이러한 작업자들은 각 개별 사용자의 관심사와 필요에 맞춤화된 상품을 제안할 수 있습니다.

효과적인 상품 추천의 핵심은 협업 필터링과 콘텐츠 기반 필터링 기술을 조합하여 활용하는 것입니다. 협업 필터링은 유사한 취향을 가진 다른 사용자들이 구매하거나 좋아한 것을 기반으로 패턴을 식별하고 추천을 제공하기 위해 비슷한 사용자들의 행동을 분석합니다. 반면에 콘텐츠 기반 필터링은 상품 자체의 특성과 속성에 중점을 두어, 사용자가 이전에 관심을 보였던 항목과 유사한 특징을 공유하는 상품을 추천합니다.

다음은 “[Railway Oriented \(ROP\)](#)” 함수형 프로그래밍 스타일을 사용하여 Ruby로 상품 추천 작업자를 구현하는 간단한 예시입니다:

```
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
```



예제에서 사용된 Ruby 함수형 프로그래밍 스타일은 F#와 Rust의 영향을 받았습니다. 이 기법에 대한 자세한 내용은 제 친구 Chad Wooley가 GitLab에 작성한 [기법 설명](#)에서 확인할 수 있습니다.

이 예제에서 `ProductRecommendationWorker`는 사용자를 입력으로 받아 값 객체를 함수형 단계들의 체인으로 전달하여 개인화된 상품 추천을 생성합니다. 각 단계를 자세히 살펴보겠습니다:

1. `ValidateUser.validate`: 이 단계에서는 사용자가 개인화된 추천을 받을 수 있는 유효한 자격이 있는지 확인합니다. 사용자가 존재하는지, 활성 상태인지, 그리고 추천을 생성하는 데 필요한 데이터가 있는지 확인합니다. 유효성 검사가 실패하면 오류 결과가 반환되고 체인이 단락됩니다.
2. `AnalyzeCurrentSession.analyze`: 사용자가 유효한 경우, 이 단계에서는 맥락 정보를 수집하기 위해 사용자의 현재 브라우징 세션을 분석합니다. 사용자의 현재 관심사와 의도를 파악하기 위해 최근에 본 상품, 검색 쿼리, 장바구니 내용과 같은 최근 상호작용을 살펴봅니다.
3. `CollaborativeFilter.filter`: _유사한 사용자들의 행동_을 활용하여, 이 단계에서는 협업 필터링 기법을 적용해 사용자가 관심을 가질 만한 상품을 식별합니다. 구매 기록, 평점, 사용자-상품 상호작용 등의 요소를 고려하여 후보 추천 세트를 생성합니다.
4. `ContentBasedFilter.filter`: 이 단계에서는 콘텐츠 기반 필터링을 적용하여 후보 추천을 더욱 정제합니다. _사용자의 선호도와 과거 데이터_를 후보 상품의 속성 및 특성과 비교하여 가장 관련성이 높은 항목을 선택합니다.
5. `ProductSelector.select`: 마지막으로, 이 단계에서는 관련성 점수, 인기도 또는 기타 비즈니스 규칙과 같은 미리 정의된 기준에 따라 필터링된 추천 목록에서 상위 N개의 상품을 선택합니다. 선택된 상품들이 최종 개인화 추천으로 반환됩니다.

여기서 함수형 Ruby 프로그래밍 스타일을 사용하는 것의 장점은 이러한 단계들을 명확하고 간결한 방식으로 연결할 수 있다는 것입니다. 각 단계는 특정 작업에 집중하고 성공(ok) 또는 오류(err)가 될 수 있는 `Result` 객체를 반환합니다. 어떤 단계에서 오류가 발생하면 체인이 단락되고 오류가 최종 결과로 전파됩니다.

마지막의 `case` 문에서는 최종 결과에 대해 패턴 매칭을 수행합니다. 결과가 오류(`ProductRecommendationError`)인 경우, Honeybadger와 같은 도구를 사용하여 모니터링 및 디버깅을 위해 오류를 기록합니다. 결과가 성공

(**ProductRecommendations**)인 경우, Wisper 발행/구독 라이브러리를 사용하여 사용자와 생성된 추천을 포함한 :**new_recommendations** 이벤트를 브로드캐스트합니다.

함수형 프로그래밍 기법을 활용함으로써 모듈화되고 유지보수가 용이한 상품 추천 워커를 만들 수 있습니다. 각 단계는 독립적이며 전체 흐름에 영향을 주지 않고도 쉽게 테스트, 수정 또는 교체할 수 있습니다. 패턴 매칭과 **Result** 클래스의 사용은 오류를 우아하게 처리하고 어떤 단계에서 문제가 발생하면 워커가 빠르게 실패하도록 보장합니다.

물론 이는 단순화된 예제이며, 실제 상황에서는 이커머스 플랫폼과의 통합, 엣지 케이스 처리, 추천 알고리즘의 구현까지 다뤄야 할 것입니다. 하지만 문제를 작은 단계들로 분해하고 함수형 프로그래밍 기법을 활용하는 핵심 원칙은 동일하게 적용됩니다.

사기 탐지

다음은 Ruby에서 동일한 철도 지향 프로그래밍(ROP) 스타일을 사용하여 사기 탐지 워커를 구현하는 간단한 예제입니다:

```
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      end
13    end
14  end
```

```
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
```

FraudDetection 클래스는 주어진 거래에 대한 사기 탐지 상태를 캡슐화하는 _값 객체입니다. 이는 다양한 위험 요소를 기반으로 거래와 관련된 사기 위험을 분석하고 평가하기 위한 구조화된 방법을 제공합니다.

```
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
```

FraudDetection 클래스는 다음과 같은 속성을 가지고 있습니다:

- transaction: 사기 분석 대상이 되는 거래에 대한 참조입니다.
- risk_factors: 거래와 관련된 위험 요소들을 저장하는 배열입니다. 각 위험 요소는 해시로 표현되며, 키는 위험 요소에 대한 설명이고 값은 해당 위험 요소와 관련된 사기 확률입니다.

add_risk_factor 메서드는 risk_factors 배열에 위험 요소를 추가할 수 있게 해줍니다. 이 메서드는 두 개의 매개변수를 받습니다: 위험 요소를 설명하는 문자열인 description과 해당 위험 요소와 관련된 사기 확률을 나타내는 실수 값인 probability입니다. 간단한 타입 검사를 위해 case..in 조건문을 사용합니다.

체인의 마지막에서 확인될 `high_risk?` 술어 메서드는 `fraud_probability`(모든 위험 요소의 확률을 합산하여 계산됨)를 `RISK_THRESHOLD`와 비교합니다.

`FraudDetection` 클래스는 거래의 사기 탐지를 관리하는 깔끔하고 캡슐화된 방법을 제공합니다. 각각의 설명과 확률을 가진 여러 위험 요소들을 추가할 수 있으며, 계산된 사기 확률을 기반으로 거래가 고위험으로 간주되는지 판단하는 메서드를 제공합니다. 이 클래스는 더 큰 사기 탐지 시스템에 쉽게 통합될 수 있으며, 다양한 구성 요소들이 협력하여 사기성 거래의 위험을 평가하고 완화할 수 있습니다.

마지막으로, 이 책이 결국 AI를 사용한 프로그래밍에 관한 것이므로, 제가 만든 `Raix` 라이브러리의 `ChatCompletion` 모듈을 활용한 `CheckCustomerHistory` 클래스의 구현 예시를 보여드리겠습니다:

```
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
```

이 예시에서 `CheckCustomerHistory`는 AI 모델에게 시스템 지시어를 통해 사기 징후를 탐지하기 위한 고객의 거래 내역을 분석하는 방법에 대한 구체적인 지침을 제공하는 `INSTRUCTION` 상수를 정의합니다.

`self.check` 메서드는 `fraud_detection` 객체로 `CheckCustomerHistory`의 새 인스턴스를 초기화하고 `call` 메서드를 호출하여 고객 내역 분석을 수행하는 클래스 메서드입니다.

`call` 메서드 내에서는 고객의 거래 내역을 검색하여 AI 모델에 전달될 트랜스 크립트 형식으로 포맷합니다. AI 모델은 제공된 지침을 바탕으로 거래 내역을 분석하고 분석 결과의 요약본을 반환합니다.

이 분석 결과는 `fraud_detection` 객체에 추가되며, 업데이트된 `fraud_detection` 객체는 성공적인 `Result`로 반환됩니다.

`ChatCompletion` 모듈을 활용함으로써, `CheckCustomerHistory` 클래스는 AI의 능력을 활용하여 고객의 거래 내역을 분석하고 잠재적인 사기 징후를 식별할 수 있습니다. AI 모델이 시간이 지남에 따라 새로운 패턴과 이상 징후를 학습하고 적응할 수 있기 때문에, 더욱 정교하고 적응적인 사기 탐지 기술이 가능해집니다.

업데이트된 `FraudDetectionWorker`와 `CheckCustomerHistory` 클래스는 AI 워커들이 원활하게 통합되어 지능적 분석과 의사결정 능력으로 사기 탐지 프로세스를 향상시키는 방법을 보여줍니다.

고객 감정 분석

다음은 고객 감정 분석 워커를 구현하는 방법에 대한 또 다른 유사한 예시입니다. 이제 이러한 프로그래밍 스타일이 어떻게 작동하는지 이해하고 계실 테니, 이번에는 설명을 많이 줄이겠습니다:

```
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
```

이 예시에서 `CustomerSentimentAnalysisWorker`의 단계에는 피드백 전처리 (예: 노이즈 제거, 토큰화), 전반적인 감정(긍정, 부정 또는 중립)을 판단하기 위

한 감정 분석 수행, 주요 문구와 주제 추출, 트렌드와 패턴 식별, 그리고 분석을 기반으로 한 실행 가능한 인사이트 생성이 포함됩니다.

의료 분야 응용

의료 분야에서 AI 워커는 의료 전문가와 연구자들의 다양한 업무를 지원하여 환자 치료 결과 개선과 의학적 발견을 가속화하는 데 도움을 줄 수 있습니다. 예시는 다음과 같습니다:

환자 접수

AI 워커는 다양한 업무를 자동화하고 지능적인 지원을 제공함으로써 환자 접수 과정을 간소화할 수 있습니다.

예약 일정 관리: AI 워커는 환자의 선호도, 가용성, 의료적 필요의 긴급성을 이해하여 예약 일정을 관리할 수 있습니다. 대화형 인터페이스를 통해 환자와 상호작용하면서 예약 과정을 안내하고, 환자의 요구사항과 의료 제공자의 가용성을 기반으로 가장 적합한 예약 시간을 찾을 수 있습니다.

의료 이력 수집: 환자 접수 과정에서 AI 워커는 환자의 의료 이력을 수집하고 문서화하는 데 도움을 줄 수 있습니다. 환자와의 대화형 상호작용을 통해 과거 병력, 복용 약물, 알레르기, 가족력에 대한 관련 질문을 할 수 있습니다. AI 워커는 자연어 처리 기술을 사용하여 수집된 정보를 해석하고 구조화하여 환자의 전자 건강 기록에 정확하게 기록됨을 보장합니다.

증상 평가 및 분류: AI 워커는 환자의 현재 증상, 지속 기간, 심각도 및 관련 요인들에 대해 질문함으로써 초기 증상 평가를 수행할 수 있습니다. 의학 지식 데이터베이스와 머신 러닝 모델을 활용하여 제공된 정보를 분석하고, 예비 감별 진단을 생성하거나 의료 제공자와의 상담 예약 또는 자가 관리 조치와 같은 적절한 다음 단계를 추천할 수 있습니다.

보험 확인: AI 워커는 환자 접수 과정에서 보험 확인을 지원할 수 있습니다. API 또는 웹 서비스를 통해 환자의 보험 정보를 수집하고 보험사와 소통하여

보장 자격과 혜택을 확인할 수 있습니다. 이러한 자동화는 보험 확인 과정을 간소화하여 행정적 부담을 줄이고 정확한 정보 수집을 보장합니다.

환자 교육 및 안내: AI 워커는 환자의 특정 의료 상태나 예정된 시술에 기반하여 관련 교육 자료와 안내를 제공할 수 있습니다. 개인화된 콘텐츠를 전달하고, 일반적인 질문에 답변하며, 예약 전 준비사항, 약물 복용 지침 또는 치료 후 관리에 대한 안내를 제공할 수 있습니다. 이는 환자가 의료 여정 전반에 걸쳐 정보를 제공받고 참여할 수 있도록 돕습니다.

AI 워커를 환자 접수에 활용함으로써 의료 기관은 효율성을 높이고, 대기 시간을 줄이며, 전반적인 환자 경험을 개선할 수 있습니다. 이러한 워커들은 일상적인 업무를 처리하고, 정확한 정보를 수집하며, 개인화된 지원을 제공하여 의료 전문가들이 환자에게 양질의 의료 서비스를 제공하는 데 집중할 수 있도록 합니다.

환자 위험 평가

AI 워커는 다양한 데이터 소스를 분석하고 고급 분석 기술을 적용하여 환자 위험을 평가하는 데 중요한 역할을 할 수 있습니다.

데이터 통합: AI 워커는 전자 건강 기록(EHR), 의료 영상, 검사 결과, 웨어러블 기기, 건강의 사회적 결정요인과 같은 다양한 출처의 환자 데이터를 수집하고 이해할 수 있습니다. 이 정보를 종합적인 환자 프로필로 통합함으로써, AI 워커는 환자의 건강 상태와 위험 요인에 대한 전체적인 관점을 제공할 수 있습니다.

위험도 분류: AI 워커는 예측 모델을 사용하여 개인의 특성과 건강 데이터를 기반으로 환자를 다양한 위험 범주로 분류할 수 있습니다. 이러한 위험도 분류를 통해 의료 제공자는 더 즉각적인 주의나 개입이 필요한 환자를 우선순위화할 수 있습니다. 예를 들어, 특정 질환에 대해 고위험군으로 식별된 환자는 더 면밀한 모니터링, 예방 조치 또는 조기 개입을 위해 표시될 수 있습니다.

개인화된 위험 프로필: AI 워커는 각 환자에 대해 위험 점수에 기여하는 특정 요인들을 강조하는 개인화된 위험 프로필을 생성할 수 있습니다. 이러한 프로

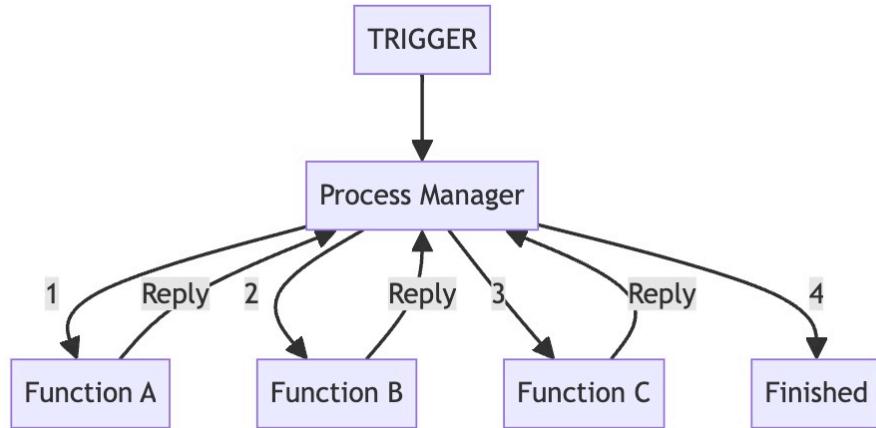
필에는 환자의 생활 방식, 유전적 소인, 환경적 요인, 건강의 사회적 결정요인에 대한 통찰이 포함될 수 있습니다. 위험 요인에 대한 상세한 분석을 제공함으로써, AI 워커는 의료 제공자가 개별 환자의 필요에 맞는 예방 전략과 치료 계획을 수립하는 데 도움을 줄 수 있습니다.

지속적인 위험 모니터링: AI 워커는 환자 데이터를 지속적으로 모니터링하고 실시간으로 위험 평가를 업데이트할 수 있습니다. 활력 징후, 검사 결과 또는 약물 복용 순응도의 변화와 같은 새로운 정보가 입수되면, AI 워커는 위험 점수를 재계산하고 의료 제공자에게 중요한 변화를 알릴 수 있습니다. 이러한 사전 예방적 모니터링을 통해 적시에 개입하고 환자 치료 계획을 조정할 수 있습니다.

임상 의사결정 지원: AI 워커는 위험 평가 결과를 임상 의사결정 지원 시스템에 통합하여 의료 제공자에게 근거 기반 권장 사항과 경고를 제공할 수 있습니다. 예를 들어, 특정 질환에 대한 환자의 위험 점수가 특정 임계값을 초과하는 경우, AI 워커는 임상 지침과 모범 사례를 기반으로 특정 진단 검사, 예방 조치 또는 치료 옵션을 고려하도록 의료 제공자에게 알림을 줄 수 있습니다.

이러한 워커들은 방대한 양의 환자 데이터를 처리하고, 정교한 분석을 적용하며, 임상 의사 결정을 지원하기 위한 실행 가능한 인사이트를 생성할 수 있습니다. 이는 궁극적으로 환자 치료 결과 개선, 의료 비용 절감, 그리고 향상된 인구 건강 관리로 이어집니다.

프로세스 관리자로서의 AI 워커



AI 기반 애플리케이션의 맥락에서, 워커는 Gregor Hohpe의 “Enterprise Integration Patterns” 책에서 설명된 Process Manager로 기능하도록 설계될 수 있습니다. Process Manager는 프로세스의 상태를 유지하고 중간 결과를 기반으로 다음 처리 단계를 결정하는 중앙 구성 요소입니다.

AI 워커가 Process Manager로 작동할 때, 프로세스를 초기화하는 트리거 메시지라고 하는 수신 메시지를 받습니다. 그런 다음 AI 워커는 프로세스 실행 상태를 (대화 기록으로) 유지하고, 도구 함수로 구현된 일련의 처리 단계를 통해 메시지를 처리합니다. 이러한 단계들은 순차적이거나 병렬적일 수 있으며, AI 워커의 재량에 따라 호출됩니다.



GPT-4와 같이 함수를 병렬로 실행하는 방법을 아는 AI 모델 클래스를 사용하는 경우, 워커가 여러 단계를 동시에 실행할 수 있습니다. 솔직히 말하면, 저는 직접 시도해보지는 않았으며, 제 직감으로는 결과가 다양할 수 있을 것 같습니다.

각각의 개별 처리 단계 후에는 제어권이 AI 워커로 반환되어, 현재 상태와 얻은 결과를 기반으로 다음 처리 단계를 결정할 수 있습니다.

트리거 메시지 저장하기

제 경험상, 트리거 메시지를 데이터베이스 기반 객체로 구현하는 것이 현명합니다. 이렇게 하면 각 프로세스 인스턴스가 고유한 기본 키로 식별되고, AI의 대화 기록을 포함한 실행 관련 상태를 저장할 수 있는 공간이 생깁니다.

예를 들어, 다음은 사용자 계정 변경 요청을 나타내는 Olympia의 `AccountChange` 모델 클래스를 단순화한 버전입니다.

```
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
```

`AccountChange` 클래스는 계정 변경 요청을 처리하는 프로세스를 시작하는 트리거 메시지 역할을 합니다. `create` 트랜잭션이 커밋을 완료한 후 Olympia의 `Wisper` 기반 발행-구독 하위 시스템으로 브로드캐스트되는 방식에 주목해보세요.

이처럼 트리거 메시지를 데이터베이스에 저장하면 각 계정 변경 요청에 대한 영구적인 기록이 생성됩니다. `AccountChange` 클래스의 각 인스턴스에는 고유한 기본 키가 할당되어 개별 요청을 쉽게 식별하고 추적할 수 있습니다. 이는

특히 감사 로깅 목적에서 유용한데, 시스템이 요청된 시점, 요청된 변경 사항, 각 요청의 현재 상태를 포함한 모든 계정 변경 사항의 이력을 유지할 수 있기 때문입니다.

주어진 예시에서 `AccountChange` 클래스는 요청된 변경 사항의 세부 정보를 담는 `description`, 요청의 현재 상태(예: `requested`, `complete`, `requires_human_review`)를 나타내는 `state`, 요청과 관련된 AI의 대화 내용을 저장하는 `transcript` 등의 필드를 포함합니다. `description` 필드는 AI와의 첫 번째 채팅 완성을 시작하는 데 사용되는 실제 프롬프트입니다. 이러한 데이터를 저장함으로써 가치 있는 컨텍스트를 제공하고 계정 변경 프로세스를 더 잘 추적하고 분석할 수 있습니다.

데이터베이스에 트리거 메시지를 저장하면 강력한 오류 처리와 복구가 가능해집니다. 계정 변경 요청 처리 중에 오류가 발생하면, 시스템은 해당 요청을 실패로 표시하고 사람의 개입이 필요한 상태로 전환합니다. 이를 통해 요청이 유실되거나 잊혀지지 않도록 하며, 모든 문제를 적절히 처리하고 해결할 수 있습니다.

프로세스 관리자로서의 AI 워커는 중앙 제어점을 제공하고 강력한 프로세스 보고 및 디버깅 기능을 가능하게 합니다. 하지만 애플리케이션의 모든 워크플로우 시나리오에 AI 워커를 프로세스 관리자로 사용하는 것은 과도할 수 있다는 점을 유의해야 합니다.

AI 워커를 애플리케이션 아키텍처에 통합하기

AI 워커를 애플리케이션 아키텍처에 통합할 때는 AI 워커와 다른 애플리케이션 구성 요소 간의 원활한 통합과 효과적인 통신을 보장하기 위해 여러 기술적 고려 사항을 다뤄야 합니다. 이 섹션에서는 인터페이스 설계, 데이터 흐름 처리, AI 워커의 수명 주기 관리에 대한 주요 측면을 살펴봅니다.

명확한 인터페이스와 통신 프로토콜 설계

AI 워커와 다른 애플리케이션 구성 요소 간의 원활한 통합을 위해서는 명확한 인터페이스와 통신 프로토콜을 정의하는 것이 중요합니다. 다음과 같은 접근 방식을 고려해보세요:

API 기반 통합: RESTful 엔드포인트나 GraphQL 스키마와 같은 잘 정의된 API를 통해 AI 워커의 기능을 노출합니다. 이를 통해 다른 구성 요소들이 표준 HTTP 요청과 응답을 사용하여 AI 워커와 상호 작용할 수 있습니다. API 기반 통합은 AI 워커와 소비 구성 요소 간의 명확한 계약을 제공하여 통합 지점의 개발, 테스트, 유지 관리를 더 쉽게 만듭니다.

메시지 기반 통신: 메시지 큐나 발행-구독 시스템과 같은 메시지 기반 통신 패턴을 구현하여 AI 워커와 다른 구성 요소 간의 비동기 상호 작용을 가능하게 합니다. 이 접근 방식은 AI 워커를 애플리케이션의 나머지 부분과 분리하여 더 나은 확장성, 장애 허용성, 느슨한 결합을 가능하게 합니다. 메시지 기반 통신은 AI 워커가 수행하는 처리가 시간이 많이 걸리거나 리소스를 많이 사용하는 경우에 특히 유용한데, 이는 AI 워커가 작업을 완료할 때까지 기다리지 않고도 애플리케이션의 다른 부분이 계속 실행될 수 있게 해주기 때문입니다.

이벤트 기반 아키텍처: 특정 조건이 충족될 때 AI 워커를 활성화하는 이벤트와 트리거를 중심으로 시스템을 설계합니다. AI 워커는 관련 이벤트를 구독하고 그에 따라 반응하여 이벤트가 발생할 때 지정된 작업을 수행할 수 있습니다. 이벤트 기반 아키텍처는 실시간 처리를 가능하게 하고 AI 워커가 필요할 때만 호출되도록 하여 불필요한 리소스 소비를 줄입니다. 이 접근 방식은 AI 워커가 특정 작업이나 애플리케이션 상태 변화에 응답해야 하는 시나리오에 적합합니다.

데이터 흐름과 동기화 처리

AI 워커를 애플리케이션에 통합할 때는 AI 워커와 다른 구성 요소 간의 원활한 데이터 흐름을 보장하고 데이터 일관성을 유지하는 것이 중요합니다. 다음과

같은 측면을 고려해보세요:

데이터 준비: AI 워커에 데이터를 공급하기 전에 입력 데이터의 정제, 형식 지정, 변환과 같은 다양한 데이터 준비 작업을 수행해야 할 수 있습니다. AI 워커가 효과적으로 처리할 수 있도록 하는 것뿐만 아니라, 워커가 최악의 경우 방해가 되고 최선의 경우에도 쓸모없다고 여길 수 있는 정보에 토큰을 낭비하지 않도록 해야 합니다. 데이터 준비에는 노이즈 제거, 누락된 값 처리, 데이터 타입 변환과 같은 작업이 포함될 수 있습니다.

데이터 영속성: AI 워커를 오가는 데이터를 어떻게 저장하고 유지할 것인지 고려해보세요. 데이터 볼륨, 쿼리 패턴, 확장성과 같은 요소들을 고려하세요. 감사나 디버깅 목적으로 AI의 “사고 과정”을 반영하는 대화 내용을 유지해야 하나요, 아니면 결과만 기록하는 것으로 충분한가요?

데이터 검색: 작업자가 필요로 하는 데이터를 얻기 위해서는 데이터베이스 쿼리, 파일 읽기, 또는 외부 API 접근이 필요할 수 있습니다. 자연 시간과 AI 작업자들이 최신 데이터에 어떻게 접근할 것인지 고려해야 합니다. 데이터베이스에 대한 전체 접근 권한이 필요한지, 아니면 수행하는 작업에 따라 접근 범위를 좁게 정의해야 하는지 결정해야 합니다. 확장성은 어떻게 할까요? 성능을 개선하고 기본 데이터 소스의 부하를 줄이기 위한 캐싱 메커니즘을 고려하세요.

데이터 동기화: AI 작업자를 포함한 여러 구성 요소가 공유 데이터에 접근하고 수정할 때, 데이터 일관성을 유지하기 위해 적절한 동기화 메커니즘을 구현하는 것이 중요합니다. 낙관적 또는 비관적 잠금과 같은 데이터베이스 잠금 전략은 충돌을 방지하고 데이터 무결성을 보장하는 데 도움이 될 수 있습니다. 관련 데이터 작업을 그룹화하고 원자성, 일관성, 격리성 및 지속성(ACID) 속성을 유지하기 위한 트랜잭션 관리 기술을 구현하세요.

오류 처리 및 복구: 데이터 흐름 프로세스 중에 발생할 수 있는 데이터 관련 문제를 처리하기 위한 강력한 오류 처리 및 복구 메커니즘을 구현하세요. 예외를 우아하게 처리하고 디버깅에 도움이 되는 의미 있는 오류 메시지를 제공하세요. 일시적인 실패나 네트워크 중단을 처리하기 위한 재시도 메커니즘과 대체 전략을 구현하세요. 데이터 손상이나 손실의 경우를 대비한 명확한 데이터 복구

및 복원 절차를 정의하세요.

데이터 흐름과 동기화 메커니즘을 신중하게 설계하고 구현함으로써, AI 작업자들이 정확하고 일관된 최신 데이터에 접근할 수 있도록 보장할 수 있습니다. 이를 통해 작업자들이 효과적으로 작업을 수행하고 신뢰할 수 있는 결과를 생산할 수 있습니다.

AI 작업자의 생명주기 관리

AI 작업자를 초기화하고 구성하기 위한 표준화된 프로세스를 개발하세요. 저는 모델 이름, 시스템 지시사항, 함수 정의 등의 설정을 표준화하는 프레임워크를 선호합니다. 배포와 확장을 용이하게 하기 위해 초기화 프로세스가 자동화되고 재현 가능하도록 보장하세요.

AI 작업자의 상태와 성능을 추적하기 위한 포괄적인 모니터링 및 로깅 메커니즘을 구현하세요. 리소스 사용률, 처리 시간, 오류율, 처리량과 같은 지표를 수집하세요. ELK 스택(Elasticsearch, Logstash, Kibana)과 같은 중앙 집중식 로깅 시스템을 사용하여 여러 AI 작업자의 로그를 집계하고 분석하세요.

AI 작업자 아키텍처에 내결함성과 복원력을 구축하세요. 실패나 예외를 우아하게 처리하기 위한 오류 처리 및 복구 메커니즘을 구현하세요. 대규모 언어 모델은 아직 최첨단 기술입니다; 제공업체들은 예상치 못한 시간에 자주 다운됩니다. 연쇄 실패를 방지하기 위해 재시도 메커니즘과 회로 차단기를 사용하세요.

AI 작업자의 구성 가능성과 오케스트레이션

AI 작업자 아키텍처의 주요 장점 중 하나는 복잡한 문제를 해결하기 위해 여러 AI 작업자를 결합하고 조율할 수 있는 구성 가능성입니다. 더 큰 작업을 전문화된 AI 작업자가 처리하는 더 작고 관리하기 쉬운 하위 작업으로 분해함으로써, 강력하고 유연한 시스템을 만들 수 있습니다. 이 섹션에서는 “다수의” AI 작업자를 구성하고 조율하는 다양한 접근 방식을 살펴보겠습니다.

다단계 워크플로우를 위한 AI 작업자 체이닝

많은 시나리오에서 복잡한 작업은 한 AI 작업자의 출력이 다음 작업자의 입력이 되는 일련의 순차적 단계로 분해될 수 있습니다. 이러한 AI 작업자의 체이닝은 다단계 워크플로우 또는 파이프라인을 생성합니다. 체인의 각 AI 작업자는 특정 하위 작업에 집중하며, 최종 출력은 모든 작업자의 결합된 노력의 결과입니다.

Ruby on Rails 애플리케이션 컨텍스트에서 사용자 생성 콘텐츠를 처리하는 예를 살펴보겠습니다. 워크플로우는 다음 단계를 포함하며, 실제로는 각각이 이런 방식으로 분해할 만큼 복잡하지 않을 수 있지만, 예제를 이해하기 쉽게 만들어줍니다:

1. **텍스트 정리:** HTML 태그 제거, 텍스트를 소문자로 변환, 유니코드 정규화를 처리하는 AI 작업자.
2. **언어 감지:** 정리된 텍스트의 언어를 식별하는 AI 작업자.
3. **감정 분석:** 감지된 언어를 기반으로 텍스트의 감정(긍정, 부정 또는 중립)을 결정하는 AI 작업자.
4. **콘텐츠 분류:** 자연어 처리 기술을 사용하여 텍스트를 미리 정의된 카테고리로 분류하는 AI 작업자.

다음은 Ruby를 사용하여 이러한 AI 작업자들을 체이닝하는 매우 단순화된 예시입니다:

```
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
```

이 예제에서 ContentProcessor 클래스는 원시 텍스트로 초기화되며 process 메서드에서 AI 워커들을 함께 연결합니다. 각 AI 워커는 자신의 특정 작업을 수행하고 그 결과를 체인의 다음 워커에게 전달합니다. 최종 출력은 정제된 텍스트, 감지된 언어, 감정 분석 결과, 그리고 콘텐츠 카테고리를 포함하는 해시입니다.

독립적인 AI 워커를 위한 병렬 처리

이전 예제에서 AI 워커들은 순차적으로 연결되어 있어서, 각 워커가 텍스트를 처리하고 그 결과를 다음 워커에게 전달했습니다. 하지만 동일한 입력에 대해 독립적으로 작동할 수 있는 여러 AI 워커가 있다면, 이들을 병렬로 처리하여 워크플로우를 최적화할 수 있습니다.

주어진 시나리오에서 TextCleanupWorker가 텍스트 정제를 수행한 후에는 LanguageDetectionWorker, SentimentAnalysisWorker, 그리고

CategorizationWorker가 모두 정제된 텍스트를 독립적으로 처리할 수 있습니다. 이러한 워커들을 병렬로 실행함으로써 전체 처리 시간을 잠재적으로 줄이고 워크플로우의 효율성을 향상시킬 수 있습니다.

Ruby에서 병렬 처리를 구현하기 위해 스레드나 비동기 프로그래밍과 같은 동시성 기법을 활용할 수 있습니다. 다음은 스레드를 사용하여 마지막 세 개의 워커를 병렬로 처리하도록 ContentProcessor 클래스를 수정하는 예제입니다:

```
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
```

이 최적화된 버전에서는 `concurrent-ruby` 라이브러리를 사용하여 독립적인 AI 워커들을 위한 `Concurrent::Future` 객체들을 생성합니다. `Future`는 별도의 스레드에서 비동기적으로 수행될 연산을 나타냅니다.

텍스트 정제 단계 이후, 우리는 세 개의 `Future` 객체를 생성합니다: `language_future`, `sentiment_future`, 그리고 `category_future`입니다. 각 `Future`는 해당하는 AI 워커(`LanguageDetectionWorker`, `SentimentAnalysisWorker`, 그리고 `CategorizationWorker`)를 별도의 스레드에서 실행하며, `cleaned_text`를 입력으로 전달합니다.

각 `Future`의 `value` 메서드를 호출함으로써, 우리는 연산이 완료될 때까지 기다리고 결과를 받아옵니다. `value` 메서드는 결과가 사용 가능할 때까지 블로킹되어, 모든 병렬 워커들이 처리를 완료하기를 기다립니다.

마지막으로, 원래 예제와 마찬가지로 정제된 텍스트와 병렬 워커들의 결과를 포함하는 출력 해시를 구성합니다.

독립적인 AI 워커들을 병렬로 처리함으로써, 순차적으로 실행하는 것에 비해 전체 처리 시간을 잠재적으로 줄일 수 있습니다. 이러한 최적화는 특히 시간이 많이 소요되는 작업을 처리하거나 대량의 데이터를 처리할 때 유용합니다.

하지만, 실제 성능 향상은 각 워커의 복잡도, 사용 가능한 시스템 리소스, 스레드 관리의 오버헤드 등 다양한 요소에 따라 달라진다는 점을 유의해야 합니다. 특정 사용 사례에 대한 최적의 병렬 처리 수준을 결정하기 위해서는 항상 벤치마크와 프로파일링을 수행하는 것이 좋은 방법입니다.

또한, 병렬 처리를 구현할 때는 워커들 간의 공유 리소스나 의존성에 주의를 기울여야 합니다. 워커들이 충돌이나 경쟁 조건 없이 독립적으로 작동할 수 있도록 해야 합니다. 의존성이나 공유 리소스가 있는 경우, 데이터 무결성을 유지하고 교착 상태나 일관성 없는 결과와 같은 문제를 피하기 위해 적절한 동기화 메커니즘을 구현해야 할 수 있습니다.

Ruby의 전역 인터프리터 잠금과 비동기 처리

Ruby에서 비동기 스레드 기반 처리를 고려할 때 전역 인터프리터 잠금 (GIL)의 영향을 이해하는 것이 중요합니다.

GIL은 Ruby 인터프리터에서 멀티코어 프로세서에서도 한 번에 하나의 스레드만 Ruby 코드를 실행할 수 있도록 보장하는 메커니즘입니다. 이는 Ruby 프로세스 내에서 여러 스레드를 생성하고 관리할 수 있지만, 주어진 시점에서 실제로 Ruby 코드를 실행할 수 있는 스레드는 하나뿐이라는 것을 의미합니다.

GIL은 Ruby 인터프리터의 구현을 단순화하고 Ruby의 내부 데이터 구조에 대한 스레드 안전성을 제공하도록 설계되었습니다. 하지만 이로 인해 Ruby 코드의 진정한 병렬 실행 가능성이 제한됩니다.

Ruby에서 `concurrent-ruby` 라이브러리나 내장된 `Thread` 클래스와 같이 스레드를 사용할 때, 이러한 스레드들은 GIL의 제약을 받습니다. GIL은 각 스레드가 짧은 시간 동안 Ruby 코드를 실행한 후 다른 스레드로 전환하도록 허용하여, 동시 실행의 환상을 만들어냅니다.

하지만 GIL로 인해 Ruby 코드의 실제 실행은 순차적으로 이루어집니다. 한 스레드가 Ruby 코드를 실행하는 동안, 다른 스레드들은 본질적으로 일시 중지되어 GIL을 획득하고 실행할 차례를 기다립니다.

이는 Ruby에서 스레드 기반 비동기 처리가 외부 API 응답 대기(예: 제3자가 호스팅하는 대규모 언어 모델)나 파일 I/O 작업 수행과 같은 I/O 바운드

태스크에 가장 효과적이라는 것을 의미합니다. 스레드가 I/O 작업을 만나면 GIL을 해제할 수 있어, I/O가 완료되기를 기다리는 동안 다른 스레드들이 실행될 수 있습니다.

반면에 집중적인 연산이나 장시간 실행되는 AI 워커 처리와 같은 CPU 바운드 태스크의 경우, GIL은 스레드 기반 병렬 처리의 잠재적 성능 향상을 제한할 수 있습니다. 한 번에 하나의 스레드만 Ruby 코드를 실행할 수 있기 때문에, 전체 실행 시간이 순차적 처리에 비해 크게 줄어들지 않을 수 있습니다.

Ruby에서 CPU 바운드 태스크의 진정한 병렬 실행을 달성하기 위해서는 다음과 같은 대체 접근 방식을 고려해야 할 수 있습니다:

- 각각 별도의 CPU 코어에서 실행되는 여러 Ruby 프로세스를 사용하는 프로세스 기반 병렬 처리
- GIL이 없는 C나 Rust와 같은 언어에 대한 네이티브 확장이나 인터페이스를 제공하는 외부 라이브러리나 프레임워크 활용,
- 여러 머신이나 프로세스에 걸쳐 태스크를 분산하기 위한 분산 컴퓨팅 프레임워크나 메시지 큐 활용

Ruby에서 비동기 처리를 설계하고 구현할 때는 작업의 특성과 GIL이 부과하는 제한을 고려하는 것이 중요합니다. 스레드 기반 비동기 처리는 I/O 바운드 태스크에는 이점을 제공할 수 있지만, GIL의 제약으로 인해 CPU 바운드 태스크에는 큰 성능 향상을 제공하지 못할 수 있습니다.

정확도 향상을 위한 양상을 기법

양상을 기법은 시스템의 전반적인 정확도나 견고성을 향상시키기 위해 여러 AI 워커의 출력을 결합하는 것을 포함합니다. 단일 AI 워커에 의존하는 대신, 양상을 기법은 더 정보에 기반한 결정을 내리기 위해 여러 워커의 집단 지성을 활용합니다.



앙상블은 워크플로우의 각기 다른 부분이 서로 다른 AI 모델과 잘 작동할 때 특히 중요한데, 이는 여러분이 생각하는 것보다 더 흔한 상황입니다. GPT-4와 같은 강력한 모델은 성능이 낮은 오픈 소스 옵션들에 비해 비용이 매우 높으며, 애플리케이션의 모든 워크플로우 단계에서 반드시 필요한 것은 아닙니다.

다수결 투표는 흔히 사용되는 앙상블 기법으로, 여러 AI 워커가 독립적으로 동일한 입력을 처리하고 최종 출력은 다수의 합의에 따라 결정됩니다. 이러한 접근 방식은 개별 워커의 오류 영향을 줄이고 시스템의 전반적인 신뢰성을 향상시키는 데 도움이 될 수 있습니다.

감성 분석을 위해 서로 다른 모델을 사용하거나 다른 컨테스트가 제공된 세 개의 AI 워커가 있는 예를 살펴보겠습니다. 다수결 투표를 사용하여 이들의 출력을 결합함으로써 최종 감성 예측을 결정할 수 있습니다.

```
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**

이 예시에서 `SentimentAnalysisEnsemble` 클래스는 텍스트로 초기화되며 세 가지 다른 감성 분석 AI 워커를 호출합니다. `analyze` 메서드는 각 워커로부터 예측을 수집하고 `group_by`와 `max_by` 메서드를 사용하여 다수결 감성을 결정합니다. 최종 출력은 워커 양상블에서 가장 많은 투표를 받은 감성입니다.



양상블은 분명히 병렬 처리 실험이 가치 있는 경우입니다.

AI 워커의 동적 선택과 호출

대부분의 경우, 호출할 특정 AI 워커는 런타임 조건이나 사용자 입력에 따라 달라질 수 있습니다. AI 워커의 동적 선택과 호출을 통해 시스템에 유연성과 적응성을 제공할 수 있습니다.



단일 AI 워커에 많은 기능을 넣고 이를 호출하는 방법을 설명하는 복잡한 프롬프트를 포함하고 싶은 유혹을 느낄 수 있습니다. 하지만 이런 유혹은 피하시기 바랍니다. 이 장에서 논의하는 접근 방식을 “다중 워커”라고 부르는 이유 중 하나는 더 큰 목적을 위해 각자의 작은 작업을 수행하는 많은 전문화된 워커를 갖는 것이 바람직하다는 것을 상기시키기 위해서입니다.

예를 들어, 서로 다른 AI 워커가 다양한 유형의 사용자 쿼리를 처리하는 챗봇 애플리케이션을 고려해보세요. 사용자의 입력을 기반으로 애플리케이션은 쿼리를 처리하기 위한 적절한 AI 워커를 동적으로 선택합니다.

```
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
```

이 예시에서 ChatbotController는 process_query 액션을 통해 사용자 쿼리를 받습니다. 먼저 QueryClassifierWorker를 사용하여 쿼리의 유형을 결정합니다. 분류된 쿼리 유형에 따라 컨트롤러는 적절한 AI 워커를 동적으로 선택하여 응답을 생성합니다. 이러한 동적 선택을 통해 챗봇은 다양한 유형의 쿼리를 처리하고 관련 AI 워커에게 전달할 수 있습니다.



`QueryClassifierWorker`의 작업은 비교적 단순하고 많은 컨텍스트나 함수 정의가 필요하지 않기 때문에, [mistralai/mixtral-8x7b-instruct:nitro](#)와 같은 초고속 소형 LLM을 사용하여 구현할 수 있습니다. 이는 많은 작업에서 GPT-4 수준에 근접한 성능을 보이며, 이 글을 쓰는 시점에서 Groq는 초당 444 토큰이라는 놀라운 처리량으로 이를 제공할 수 있습니다.

전통적인 NLP와 LLM의 결합

대규모 언어 모델(LLM)이 자연어 처리(NLP)분야에 혁명을 일으키고 광범위한 작업에서 타의 추종을 불허하는 다재다능함과 성능을 제공하고 있지만, 모든 문제에 대해 항상 가장 효율적이거나 비용 효율적인 해결책은 아닙니다. 많은 경우에 전통적인 NLP 기술과 LLM을 결합하면 특정 NLP 과제를 해결하는 데 있어 더욱 최적화되고, 목적에 맞으며, 경제적인 접근 방식을 만들 수 있습니다.

LLM을 NLP의 스위스 아미 나이프라고 생각해보세요—믿을 수 없을 만큼 다재다능하고 강력하지만, 모든 작업에 대해 반드시 최고의 도구는 아닙니다. 때로는 코드를 따개나 캔 오프너처럼 특정 작업을 위한 전용 도구가 더 효과적이고 효율적일 수 있습니다. 마찬가지로, 문서 군집화, 주제 식별, 분류와 같은 전통적인 NLP 기술은 종종 NLP 파이프라인의 특정 측면에 대해 더 목적에 맞고 비용 효율적인 솔루션을 제공할 수 있습니다.

전통적인 NLP 기술의 주요 장점 중 하나는 계산 효율성입니다. 단순한 통계 모델이나 규칙 기반 접근 방식에 의존하는 이러한 방법들은 LLM에 비해 훨씬 더 빠르고 낮은 계산 오버헤드로 대량의 텍스트 데이터를 처리할 수 있습니다. 이는 유사한 기사를 군집화하거나 텍스트 모음 내의 주요 주제를 식별하는 것과 같이 대량의 문서를 분석하고 조직화하는 작업에 특히 적합합니다.

더욱이, 전통적인 NLP 기술은 특히 도메인별 데이터셋으로 훈련된 경우 특정 작업에 대해 높은 정확도와 정밀도를 달성할 수 있습니다. 예를 들어, 서포트 벡터 머신(SVM)이나 나이브 베이즈와 같은 전통적인 기계 학습 알고리즘을

사용하는 잘 조정된 문서 분류기는 최소한의 계산 비용으로 문서를 미리 정의된 카테고리로 정확하게 분류할 수 있습니다.

하지만 LLM은 언어, 맥락, 추론에 대한 더 깊은 이해가 필요한 작업에서 진정한 빛을 발합니다. 일관성 있고 맥락에 맞는 텍스트를 생성하고, 질문에 답하며, 긴 문장을 요약하는 능력은 전통적인 NLP 방법으로는 따라갈 수 없습니다. LLM은 모호성, 공동참조, 관용구와 같은 복잡한 언어 현상을 효과적으로 처리할 수 있어 자연어 생성이나 이해가 필요한 작업에 매우 중요합니다.

진정한 힘은 전통적인 NLP 기술과 LLM을 결합하여 두 가지의 장점을 모두 활용하는 하이브리드 접근 방식을 만드는 데 있습니다. 문서 전처리, 군집화, 주제 추출과 같은 작업에 전통적인 NLP 방법을 사용하여 텍스트 데이터를 효율적으로 구성하고 구조화할 수 있습니다. 이렇게 구조화된 정보는 요약 생성, 질문 답변, 종합 보고서 작성과 같은 더 고급 작업을 위해 LLM에 입력될 수 있습니다.

예를 들어, 대량의 개별 트렌드 문서를 기반으로 특정 도메인에 대한 트렌드 보고서를 생성하고자 하는 사용 사례를 고려해보겠습니다. 대량의 텍스트를 처리하는 데 계산 비용이 많이 들고 시간이 오래 걸릴 수 있는 LLM에만 의존하는 대신, 다음과 같은 하이브리드 접근 방식을 사용할 수 있습니다:

1. 잠재 디리클레 할당과 같은 토픽 모델링이나 K-평균과 같은 군집화 알고리즘과 같은 전통적인 NLP 기술을 사용하여 유사한 트렌드 문서를 그룹화하고 말뭉치 내의 주요 주제와 토픽을 식별합니다.
2. 군집화된 문서와 식별된 토픽을 LLM에 입력하여, 우수한 언어 이해 및 생성 능력을 활용해 각 클러스터나 토픽에 대한 일관성 있고 유익한 요약을 생성합니다.
3. 마지막으로, LLM을 사용하여 개별 요약을 결합하고, 가장 중요한 트렌드를 강조하며, 집계된 정보를 기반으로 인사이트와 권장 사항을 제공하는 포괄적인 트렌드 보고서를 생성합니다.

이러한 방식으로 전통적인 NLP 기술과 LLM을 결합함으로써, 계산 리소스와

비용을 최적화하면서 대량의 텍스트 데이터를 효율적으로 처리하고, 의미 있는 인사이트를 추출하며, 고품질의 보고서를 생성할 수 있습니다.

NLP 프로젝트를 시작할 때, 각 작업의 구체적인 요구사항과 제약사항을 신중히 평가하고, 전통적인 NLP 방법과 LLM을 어떻게 함께 활용하여 최상의 결과를 얻을 수 있을지 고려하는 것이 매우 중요합니다. 전통적인 기법의 효율성과 정확성을 LLM의 다재다능함과 강력한 성능과 결합함으로써, 사용자와 이해 관계자들에게 가치를 전달하는 매우 효과적이고 경제적인 NLP 솔루션을 만들 수 있습니다.

도구 사용



AI 기반 애플리케이션 개발 분야에서, “도구 사용” 또는 “함수 호출”이라는 개념은 LLM이 외부 도구, API, 함수, 데이터베이스 및 기타 리소스에 연결할 수 있게 해주는 강력한 기술로 부상했습니다. 이러한 접근 방식은 단순히 텍스트를 출력하는 것 이상의 더 풍부한 동작 세트를 가능하게 하며, AI 구성 요소와 애플리케이션 생태계의 나머지 부분 사이에 더 역동적인 상호작용을 가능하게 합니다. 이 장에서 살펴보겠지만, 도구 사용을 통해 AI 모델이 구조화된 방식으로 데이터를 생성하도록 하는 옵션도 제공됩니다.

도구 사용이란 무엇인가?

도구 사용은 함수 호출이라고도 하며, 개발자가 LLM이 생성 과정 중에 상호작용할 수 있는 함수 목록을 지정할 수 있게 해주는 기술입니다. 이러한 도구들은 간단한 유틸리티 함수부터 복잡한 API나 데이터베이스 쿼리까지 다양합니다.

LLM에 이러한 도구들에 대한 접근 권한을 제공함으로써, 개발자들은 모델의 기능을 확장하고 외부 지식이나 작업이 필요한 태스크를 수행할 수 있게 만들 수 있습니다.

그림 7. 문서를 분석하는 AI 작업자를 위한 함수 정의 예시

```
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
```

도구 사용의 핵심 개념은 LLM에게 사용자의 입력이나 주어진 작업에 따라 적절한 도구를 동적으로 선택하고 실행할 수 있는 능력을 부여하는 것입니다. 모델의 사전 학습된 지식에만 의존하는 대신, 도구 사용을 통해 LLM은 외부 리소스를 활용하여 더 정확하고 관련성 있으며 실행 가능한 응답을 생성할 수 있습니다. 도구 사용은 RAG(검색 증강 생성)과 같은 기술을 구현하는 것을 훨씬 쉽게 만듭니다.

달리 명시되지 않는 한, 이 책에서는 AI 모델이 내장된 서버 측 도구에 접근할 수 없다고 가정합니다. AI가 사용할 수 있는 모든 도구는 각 API 요청에서 명시적으로 선언되어야 하며, AI가 응답에서 해당 도구를 사용하고자 할 때 실행을 위한 규정이 포함되어야 합니다.

도구 사용의 잠재력

도구 사용은 AI 기반 애플리케이션에 광범위한 가능성을 제공합니다. 다음은 도구 사용으로 달성할 수 있는 몇 가지 예시입니다:

1. **챗봇과 가상 비서**: LLM을 외부 도구와 연결함으로써, 챗봇과 가상 비서는 데이터베이스에서 정보를 검색하거나, API 호출을 실행하거나, 다른 시스템과 상호작용하는 등 더 복잡한 작업을 수행할 수 있습니다. 예를 들어, 챗봇은 CRM 도구를 사용하여 사용자의 요청에 따라 거래 상태를 변경할 수 있습니다.
2. **데이터 분석 및 인사이트**: LLM은 데이터 분석 도구나 라이브러리와 연결되어 고급 데이터 처리 작업을 수행할 수 있습니다. 이를 통해 애플

리케이션은 사용자 쿼리를 기반으로 인사이트를 생성하고, 비교 분석을 수행하거나, 데이터 기반 추천을 제공할 수 있습니다.

3. 검색 및 정보 검색: 도구 사용을 통해 LLM은 검색 엔진, 벡터 데이터베이스 또는 기타 정보 검색 시스템과 상호작용할 수 있습니다. 사용자 쿼리를 검색 쿼리로 변환함으로써, LLM은 여러 소스에서 관련 정보를 검색하고 사용자 질문에 대한 포괄적인 답변을 제공할 수 있습니다.
4. 외부 서비스와의 통합: 도구 사용을 통해 AI 기반 애플리케이션과 외부 서비스 또는 API 간의 원활한 통합이 가능합니다. 예를 들어, LLM은 날씨 API와 상호작용하여 실시간 날씨 업데이트를 제공하거나 번역 API를 통해 다국어 응답을 생성할 수 있습니다.

도구 사용 워크플로우

도구 사용 워크플로우는 일반적으로 다음 네 가지 주요 단계로 구성됩니다:

1. 요청 컨텍스트에 함수 정의 포함
2. 동적(또는 명시적) 도구 선택
3. 함수 실행
4. 선택적 원본 프롬프트 계속

각 단계를 자세히 살펴보겠습니다.

요청 컨텍스트에 함수 정의 포함

AI는 완료 요청의 일부로 제공되는 목록(일반적으로 JSON 스키마의 변형을 사용하여 함수로 정의됨)을 통해 사용 가능한 도구를 파악합니다.

도구 정의의 정확한 구문은 모델별로 다릅니다.

다음은 Claude 3에서 `get_weather` 함수를 정의하는 방법입니다:

```
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 }
```

그리고 이것이 GPT-4에 대해 동일한 함수를 정의하고 이를 tools 매개변수의 값으로 전달하는 방법입니다:

```
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     }
```

겉보기에는 특별한 이유도 없이 거의 같은데 다르다니! 정말 짜증 나네요.

함수 정의는 이름, 설명, 그리고 입력 매개변수를 지정합니다. 입력 매개변수는 열거형을 사용하여 허용 가능한 값을 제한하거나 매개변수의 필수 여부를 지정하는 등의 속성을 통해 더 자세히 정의할 수 있습니다.

실제 함수 정의 외에도, 시스템 지시어에 해당 함수를 시스템에서 사용하는 방법과 이유에 대한 설명을 포함할 수 있습니다.

예를 들어, Olympia의 웹 검색 도구에는 AI에게 언급된 도구들을 사용할 수 있다고 상기시켜주는 다음과 같은 시스템 지시어가 포함되어 있습니다:

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

상세한 설명을 제공하는 것은 도구 성능에서 가장 중요한 요소로 간주됩니다. 도구에 대한 설명에는 다음과 같은 모든 세부 사항이 포함되어야 합니다:

- 도구의 기능
- 사용해야 할 시기(그리고 사용하지 말아야 할 시기)
- 각 매개변수의 의미와 도구의 동작에 미치는 영향
- 도구 구현에 적용되는 중요한 주의사항이나 제한사항

AI에게 도구에 대한 더 많은 맥락을 제공할수록, AI는 언제 어떻게 도구를 사용할지 더 잘 결정할 수 있습니다. 예를 들어, Anthropic은 Claude 3 시리즈에서 도구당 최소 3-4문장의 설명을 권장하며, 복잡한 도구의 경우 더 많은 설명을 권장합니다.

직관적이지는 않을 수 있지만, 설명은 예시보다 더 중요한 것으로 간주됩니다. 도구 설명이나 함께 제공되는 프롬프트에 도구 사용 예시를 포함할 수 있지만, 이는 도구의 목적과 매개변수에 대한 명확하고 포괄적인 설명을 갖추는 것보다는 덜 중요합니다. 설명을 완전히 작성한 후에만 예시를 추가하세요.

다음은 Stripe 스타일의 API 함수 명세 예시입니다:

```
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 }
```



실제로 일부 모델은 중첩된 함수 명세를 처리하거나 배열, 덱셔너리 등과 같은 복잡한 출력 데이터 타입을 처리하는 데 어려움을 겪습니다. 하지만 이론적으로는 임의의 깊이를 가진 JSON 스키마 명세를 제공할 수 있어야 합니다!

동적 도구 선택

도구 정의가 포함된 채팅 완성을 실행할 때, LLM은 가장 적절한 도구를 동적으로 선택하고 각 도구에 필요한 입력 매개변수를 생성합니다.

실제로 AI가 정확히 올바른 함수를 호출하고 입력에 대한 사양을 정확히 따르는 능력은 성공할 때도 있고 실패할 때도 있습니다. temperature 하이퍼파라미터를 0.0으로 완전히 낮추면 많은 도움이 되지만, 제 경험상 여전히 가끔 오류가 발생합니다. 이러한 실패에는 환상의 함수 이름, 잘못 명명되거나 누락된 입력 매개변수가 포함됩니다. 매개변수는 JSON으로 전달되므로 때로는 잘린 JSON, 잘못된 따옴표, 또는 기타 손상된 JSON으로 인한 오류가 발생할 수 있습니다.



자가 치유 데이터 패턴은 구문 오류로 인해 실패하는 함수 호출을
자동으로 수정하는 데 도움이 될 수 있습니다.

강제(즉, 명시적) 도구 선택

일부 모델에서는 요청의 매개변수로 특정 함수의 강제 호출을 선택할 수 있습니다. 그렇지 않으면 함수를 호출할지 여부는 전적으로 AI의 재량에 달려 있습니다.

함수 호출을 강제하는 기능은 AI의 동적 선택 프로세스와 관계없이 특정 도구나 함수의 실행을 보장하고자 하는 특정 시나리오에서 매우 중요합니다. 이 기능이 중요한 이유는 다음과 같습니다:

1. **명시적 제어:** AI를 개별 구성 요소로 사용하거나 특정 시점에 특정 함수의 실행이 필요한 미리 정의된 워크플로우에서 사용할 수 있습니다. 호출을 강제함으로써 AI에게 정중하게 요청할 필요 없이 원하는 함수가 호출되도록 보장할 수 있습니다.

2. 디버깅 및 테스팅: AI 기반 애플리케이션을 개발하고 테스트할 때, 함수 호출을 강제하는 기능은 디버깅 목적으로 매우 유용합니다. 특정 함수를 명시적으로 트리거함으로써 애플리케이션의 개별 구성 요소를 분리하고 테스트할 수 있습니다. 이를 통해 함수 구현의 정확성을 확인하고, 입력 매개변수를 검증하며, 예상된 결과가 반환되는지 확인할 수 있습니다.
3. 예외 케이스 처리: AI의 동적 선택 프로세스가 실행해야 할 함수를 선택하지 않을 수 있는 예외 케이스나 특수한 시나리오가 있을 수 있으며, 이는 외부 프로세스를 기반으로 알 수 있습니다. 이러한 경우에 함수 호출을 강제하는 기능을 통해 이러한 상황을 명시적으로 처리할 수 있습니다. 애플리케이션 로직에서 AI의 재량권을 무시할 시기를 결정하는 규칙이나 조건을 정의하세요.
4. 일관성 및 재현성: 특정 순서로 실행해야 하는 특정 함수 시퀀스가 있는 경우, 호출을 강제하면 매번 동일한 시퀀스가 따라지도록 보장됩니다. 이는 금융 시스템이나 과학적 시뮬레이션과 같이 일관성과 예측 가능한 동작이 중요한 애플리케이션에서 특히 중요합니다.
5. 성능 최적화: 경우에 따라 함수 호출을 강제하면 성능 최적화로 이어질 수 있습니다. 특정 작업에 특정 함수가 필요하다는 것을 알고 있고 AI의 동적 선택 프로세스가 불필요한 오버헤드를 발생시킬 수 있다면, 선택 프로세스를 우회하고 필요한 함수를 직접 호출할 수 있습니다. 이는 자연 시간을 줄이고 애플리케이션의 전반적인 효율성을 향상시키는 데 도움이 될 수 있습니다.

요약하면, AI 기반 애플리케이션에서 함수 호출을 강제하는 기능은 명시적 제어를 제공하고, 디버깅 및 테스팅을 돋고, 예외 케이스를 처리하며, 일관성과 재현성을 보장합니다. 이는 강력한 도구이지만, 이 중요한 기능의 또 다른 측면을 논의할 필요가 있습니다.



많은 의사 결정 사용 사례에서, 우리는 모델이 항상 함수 호출을 하기를 원하며 모델이 단순히 내부 지식으로만 응답하는 것을 원하지 않을 수 있습니다. 예를 들어, 다른 작업(다국어 입력, 수학 등)에 특화된 여러 모델 간에 라우팅하는 경우, 함수 호출 모델을 사용하여 요청을 도우미 모델 중 하나에 위임하고 독립적으로 응답하지 않을 수 있습니다.

Tool Choice 매개변수

GPT-4와 함수 호출을 지원하는 다른 언어 모델들은 완성의 일부로 도구 사용이 필요한지 여부를 제어하기 위한 `tool_choice` 매개변수를 제공합니다. 이 매개변수는 세 가지 가능한 값을 가집니다:

- `auto`는 AI에게 도구를 사용하거나 단순히 응답하는 것에 대한 완전한 재량권을 줍니다
- `required`는 AI에게 응답하는 대신 도구를 반드시 호출해야 한다고 알려주지만, 도구의 선택은 AI에게 맡깁니다
- 세 번째 옵션은 강제하고 싶은 `name_of_function`의 매개변수를 설정하는 것입니다. 이에 대해서는 다음 섹션에서 자세히 다루겠습니다.



`required`로 도구 선택을 설정하면, 모델은 제공된 함수들 중에서 가장 관련성이 높은 함수를 선택하도록 강제됩니다. 이는 실제로 적합한 함수가 없는 경우에도 마찬가지입니다. 출판 시점을 기준으로, 빈 `tool_calls` 응답을 반환하거나 적절한 함수를 찾지 못했음을 알려주는 다른 방법을 사용하는 모델은 알려진 바가 없습니다.

구조화된 출력을 위한 함수 강제 실행

함수 호출을 강제할 수 있다는 것은 일반 텍스트 응답에서 직접 추출하는 대신 채팅 완성에서 구조화된 데이터를 강제로 얻을 수 있는 방법을 제공합니다.

구조화된 출력을 얻기 위해 함수를 강제하는 것이 왜 중요할까요? 간단히 말해서, LLM의 출력에서 구조화된 데이터를 추출하는 것이 매우 까다롭기 때문입니다. XML로 데이터를 요청하여 작업을 조금 더 쉽게 만들 수 있지만, 그러면 XML을 파싱해야 합니다. 그리고 AI가 “죄송하지만 블라블라한 이유로 요청하신 데이터를 생성할 수 없습니다…”라고 응답하여 XML이 누락된 경우에는 어떻게 해야 할까요?

이런 방식으로 도구를 사용할 때:

- 요청에 단일 도구만 정의하는 것이 좋습니다
- `tool_choice` 매개변수를 사용하여 함수 사용을 강제하는 것을 잊지 마세요.
- 모델이 도구에 입력을 전달한다는 점을 기억하세요. 따라서 도구의 이름과 설명은 당신의 관점이 아닌 모델의 관점에서 작성되어야 합니다.

마지막 부분은 명확성을 위해 예시가 필요합니다. 예를 들어, AI에게 사용자 텍스트에 대한 감정 분석을 요청한다고 가정해 보겠습니다. 함수의 이름은 `analyze_sentiment`가 아니라 `save_sentiment_analysis`와 같은 것이 되어야 합니다. 감정 분석을 수행하는 것은 도구가 아닌 AI입니다. 도구는 (AI의 관점에서) 단지 분석 결과를 저장하는 역할만 합니다.

다음은 명령줄에서 `curl`을 사용하여 Claude 3로 이미지의 요약을 잘 구조화된 JSON으로 기록하는 예시입니다:

```
1 curl https://api.anthropic.com/v1/messages \
2     --header "content-type: application/json" \
3     --header "x-api-key: $ANTHROPIC_API_KEY" \
4     --header "anthropic-version: 2023-06-01" \
5     --header "anthropic-beta: tools-2024-04-04" \
6     --data \
7 '{
8     "model": "claude-3-sonnet-20240229",
9     "max_tokens": 1024,
10    "tools": [
11        {
12            "name": "record_summary",
13            "description": "Record summary of image into well-structured JSON.",
14            "input_schema": {
15                "type": "object",
16                "properties": {
17                    "key_colors": {
18                        "type": "array",
19                        "items": {
20                            "type": "object",
21                            "properties": {
22                                "r": {
23                                    "type": "number",
24                                    "description": "red value [0.0, 1.0]"
25                                },
26                                "g": {
27                                    "type": "number",
28                                    "description": "green value [0.0, 1.0]"
29                                },
30                                "b": {
31                                    "type": "number",
32                                    "description": "blue value [0.0, 1.0]"
33                                }
34                            }
35                        }
36                    }
37                }
38            }
39        }
40    ]
41 }
```

```
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 }'
```

제공된 예시에서는 Anthropic의 Claude 3 모델을 사용하여 이미지의 구조화된 JSON 요약을 생성합니다. 작동 방식은 다음과 같습니다:

1. 요청 페이로드의 `tools` 배열에서 `record_summary`라는 단일 도구를 정의합니다. 이 도구는 이미지의 요약을 잘 구조화된 JSON으로 기록하는 역할을 합니다.
2. `record_summary` 도구는 JSON 출력의 예상 구조를 지정하는 `input_schema`를 가지고 있습니다. 다음과 같은 세 가지 속성을 정의합니다:

- **key_colors**: 이미지의 주요 색상을 나타내는 객체 배열입니다. 각 색상 객체는 빨강, 초록, 파랑 값(0.0에서 1.0 사이)과 사람이 읽을 수 있는 snake_case 형식의 색상 이름을 속성으로 가집니다.
 - **description**: 이미지에 대한 간단한 설명을 위한 문자열 속성으로, 1-2문장으로 제한됩니다.
 - **estimated_year**: 실제 사진으로 보이는 경우, 이미지가 촬영된 것으로 추정되는 연도를 나타내는 선택적 정수 속성입니다.
3. **messages** 배열에서는 미디어 타입과 함께 base64로 인코딩된 문자열 형태로 이미지 데이터를 제공합니다. 이를 통해 모델이 입력의 일부로 이미지를 처리할 수 있습니다.
 4. 또한 Claude에게 **record_summary** 도구를 사용하여 이미지를 설명하도록 프롬프트를 제공합니다.
 5. Claude 3 모델에 요청이 전송되면, 모델은 이미지를 분석하고 지정된 **input_schema**를 기반으로 JSON 요약을 생성합니다. 모델은 주요 색상을 추출하고, 간단한 설명을 제공하며, 해당되는 경우 이미지가 촬영된 연도를 추정합니다.
 6. 생성된 JSON 요약은 **record_summary** 도구에 매개변수로 전달되어, 이미지의 주요 특성에 대한 구조화된 표현을 제공합니다.

record_summary 도구를 잘 정의된 **input_schema**와 함께 사용함으로써, 일반 텍스트 추출에 의존하지 않고도 이미지의 구조화된 JSON 요약을 얻을 수 있습니다. 이 접근 방식은 출력이 일관된 형식을 따르도록 보장하며, 애플리케이션의 다운스트림 컴포넌트에서 쉽게 구문 분석하고 처리할 수 있도록 합니다.

AI 기반 애플리케이션에서 도구 사용의 함수 호출을 강제하고 예상 출력 구조를 지정하는 기능은 강력한 특징입니다. 이를 통해 개발자는 생성된 출력을 더 잘 제어할 수 있으며, AI가 생성한 데이터를 애플리케이션의 워크플로우에 통합하는 것이 더 쉬워집니다.

함수 실행

함수들을 정의하고 AI에게 프롬프트를 제공하면, AI는 함수 중 하나를 호출해야 한다고 결정합니다. 이제 애플리케이션 코드나 [raix-rails](#)와 같은 Ruby 젬을 사용하는 경우, 함수 호출과 그 매개변수를 애플리케이션 코드의 해당 구현으로 전달할 차례입니다.

애플리케이션 코드는 함수 실행 결과를 어떻게 처리할지 결정합니다. 람다의 한 줄 코드일 수도 있고, 외부 API를 호출하는 것일 수도 있습니다. 다른 AI 컴포넌트를 호출하거나 시스템의 나머지 부분에서 수백 또는 수천 줄의 코드를 포함할 수도 있습니다. 이는 전적으로 여러분에게 달려 있습니다.

때로는 함수 호출이 작업의 끝이지만, 결과가 AI가 계속 처리해야 할 사고 연쇄의 정보를 나타내는 경우, 애플리케이션 코드는 실행 결과를 채팅 기록에 삽입하고 AI가 처리를 계속하도록 해야 합니다.

예를 들어, 다음은 고객 서비스를 위한 지능형 워크플로우 오케스트레이션의 일부로 Olympia의 `AccountManager`가 클라이언트와 통신하기 위해 사용하는 Raix 함수 선언입니다.

```
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
```

여기서 무슨 일이 일어나고 있는지 바로 이해하기 어려울 수 있으므로, 하나씩 설명하겠습니다.

1. AccountManager 클래스는 계정 관리와 관련된 많은 함수들을 정의합니다. 이는 요금제를 변경하고, 팀 구성원을 추가하거나 제거하는 등의 작업을 수행할 수 있습니다.
2. 최상위 지침에서는 AccountManager가 notify_account_owner 함수를 사용하여 계정 변경 요청의 결과를 계정 소유자에게 알리도록 지시합니다.
3. 함수의 간단한 정의에는 다음이 포함됩니다:
 - 이름
 - 설명
 - 매개변수 message: { type: "string" }
 - 함수가 호출될 때 실행할 블록

함수 블록의 결과로 대화 기록을 업데이트한 후, chat_completion 메서드가 다시 호출됩니다. 이 메서드는 추가 처리를 위해 업데이트된 대화 기록을 AI 모델로 다시 보내는 역할을 합니다. 우리는 이 과정을 [대화 루프](#)라고 부릅니다.

AI 모델이 업데이트된 대화 기록이 포함된 새로운 채팅 완성 요청을 받으면, 이전에 실행된 함수의 결과에 접근할 수 있습니다. 이러한 결과를 분석하고, 의사 결정 과정에 통합하여, 대화의 누적된 맥락을 바탕으로 다음 응답이나 행동을 생성할 수 있습니다. 업데이트된 맥락을 기반으로 추가 함수를 실행하거나, 더 이상의 함수 호출이 필요하지 않다고 판단되면 원래 프롬프트에 대한 최종 응답을 생성할 수 있습니다.

원래 프롬프트의 선택적 계속

도구 결과를 LLM에 다시 보내고 원래 프롬프트의 처리를 계속하면, AI는 이러한 결과를 사용하여 추가 함수를 호출하거나 최종 일반 텍스트 응답을 생성합니다.



Cohere의 [Command-R](#)과 같은 일부 모델은 응답에서 사용한 특정 도구를 인용할 수 있어, 추가적인 투명성과 추적성을 제공합니다.

사용 중인 모델에 따라, 함수 호출의 결과는 자체적인 특별한 역할을 가진 대화 기록 메시지에 저장되거나 다른 구문에 반영될 수 있습니다. 하지만 중요한 점은 AI가 다음에 할 일을 결정할 때 고려할 수 있도록 해당 데이터가 대화 기록에 있어야 한다는 것입니다.



흔한 (그리고 잠재적으로 비용이 많이 드는) 오류 상황은 채팅을 계속하기 전에 함수 결과를 대화 기록에 추가하는 것을 잊는 것입니다. 그 결과, AI는 처음 함수를 호출했을 때와 본질적으로 동일한 방식으로 프롬프트를 받게 됩니다. 다시 말해, AI의 입장에서는 아직 함수를 호출하지 않은 것입니다. 그래서 다시 호출합니다. 그리고 또 다시. 당신이 중단할 때까지 계속됩니다. 컨텍스트가 너무 크지 않고 모델이 너무 비싸지 않기를 바랍니다!

도구 사용을 위한 모범 사례

도구 사용을 최대한 활용하기 위해 다음과 같은 모범 사례를 고려하세요.

설명적인 정의

각 도구와 입력 매개변수에 대해 명확하고 설명적인 이름과 설명을 제공하세요. 이는 LLM이 각 도구의 목적과 기능을 더 잘 이해하는 데 도움이 됩니다.

경험상 “이름 짓기가 어렵다”는 일반적인 통념이 여기에도 적용된다고 말씀 드릴 수 있습니다. 함수의 이름이나 설명의 표현을 바꾸는 것만으로도 LLM의 결과가 극적으로 달라지는 것을 보았습니다. 때로는 설명을 제거하는 것이 성능을 향상시키기도 합니다.

도구 결과 처리

LLM에 도구 결과를 다시 전달할 때, 잘 구조화되고 포괄적인지 확인하세요. 각 도구의 출력을 표현하기 위해 의미 있는 키와 값을 사용하세요. JSON에서 일반 텍스트에 이르기까지 다양한 형식을 실험하고 어떤 것이 가장 잘 작동하는지 확인하세요.

[결과 해석기](#)는 AI를 사용하여 결과를 분석하고 사람이 이해하기 쉬운 설명, 요약 또는 주요 내용을 제공함으로써 이 과제를 해결합니다.

오류 처리

LLM이 도구 호출에 대해 잘못되거나 지원되지 않는 입력 매개변수를 생성할 수 있는 경우를 처리하기 위한 강력한 오류 처리 메커니즘을 구현하세요. 도구 실행 중에 발생할 수 있는 모든 오류를 우아하게 처리하고 복구하세요.

AI의 매우 좋은 특징 중 하나는 오류 메시지를 이해한다는 것입니다! 이는 빠르고 간단한 마인드로 작업하는 경우, 도구 구현에서 생성된 예외를 단순히 잡아서 AI에 다시 전달하면 무슨 일이 일어났는지 알 수 있다는 것을 의미합니다!

예를 들어, 다음은 Olympia에서 구글 검색 구현의 간소화된 버전입니다:

```
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
```

Olympia에서의 Google 검색은 두 단계 과정으로 이루어집니다. 먼저 검색을 수행한 다음, 결과를 요약합니다. 어떤 종류의 실패가 발생하더라도 예외 메시지는 패키징되어 AI로 전송됩니다. 이 기술은 실질적으로 모든 지능형 오류 처리 패턴의 기초가 됩니다.

예를 들어, GoogleSearch API 호출이 503 서비스 불가용 예외로 인해 실패했다고 가정해 보겠습니다. 이는 최상위 rescue 구문으로 전달되고, 오류에 대한 설명이 함수 호출의 결과로 AI에 전송됩니다. 사용자에게 빈 화면이나 기술적인 오류를 보여주는 대신, AI는 “죄송합니다만, 현재 Google 검색 기능에 접근할 수 없습니다. 원하신다면 나중에 다시 시도해 볼 수 있습니다.”와 같은 응답을 합니다.

이것이 단순한 기발한 트릭처럼 보일 수 있지만, AI가 외부 API를 호출하고 API에 전달할 매개변수를 직접 제어하는 다른 종류의 오류를 고려해보세요. 어쩌면 AI가 이러한 매개변수를 생성하는 방식에서 실수를 했을 수도 있습니다. 외부 API의 오류 메시지가 충분히 상세하다면, 오류 메시지를 호출하는 AI에 다시 전달함으로써 AI가 이러한 매개변수를 재고하고 다시 시도할 수 있습니다. 자동으로 말이죠. 어떤 오류가 발생하더라도 말입니다.

이제 일반적인 코드에서 이러한 종류의 강력한 오류 처리를 구현하는 데 무엇이

필요할지 생각해보세요. 이는 사실상 불가능합니다.

반복적 개선

만약 LLM이 적절한 도구를 추천하지 않거나 최적화되지 않은 응답을 생성한다면, 도구 정의, 설명, 입력 매개변수를 반복적으로 수정하세요. 관찰된 동작과 원하는 결과를 바탕으로 도구 설정을 지속적으로 개선하고 향상시키세요.

1. 간단한 도구 정의로 시작하기: 명확하고 간결한 이름, 설명, 입력 매개변수로 도구를 정의하는 것부터 시작하세요. 처음에는 도구 설정을 과도하게 복잡하게 만들지 말고 핵심 기능에 집중하세요. 예를 들어, 감정 분석 결과를 저장하고 싶다면 다음과 같은 기본적인 정의로 시작하세요:

```
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. 테스트 및 관찰: 초기 도구 정의를 완료한 후, 다양한 프롬프트로 테스트하고 LLM이 도구와 어떻게 상호작용하는지 관찰하십시오. 생성된 응답의

품질과 관련성에 주의를 기울이세요. LLM이 최적화되지 않은 응답을 생성하는 경우, 도구 정의를 개선할 때입니다.

3. 설명 개선: LLM이 도구의 목적을 잘못 이해하고 있다면, 도구의 설명을 개선해 보십시오. LLM이 도구를 효과적으로 사용할 수 있도록 더 많은 맥락, 예시, 또는 설명을 제공하세요. 예를 들어, 감성 분석 도구의 설명을 분석 대상 텍스트의 감정 톤을 더 구체적으로 다루도록 업데이트할 수 있습니다:

```
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. 입력 매개변수 조정: LLM이 도구에 대해 유효하지 않거나 부적절한 입력 매개변수를 생성하는 경우, 매개변수 정의를 조정하는 것을 고려하세요. 예상되는 입력 형식을 명확히 하기 위해 더 구체적인 제약 조건, 유효성 검사 규칙 또는 예시를 추가하세요.
5. 피드백을 바탕으로 한 반복: 도구의 성능을 지속적으로 모니터링하고 사용자나 이해관계자로부터 피드백을 수집하세요. 이 피드백을 활용하여 개선이 필요한 영역을 파악하고 도구 정의를 반복적으로 개선하세요. 예를 들어, 사용자들이 분석이 풍자를 제대로 처리하지 못한다고 보고하는 경우, 설명에 관련 참고 사항을 추가할 수 있습니다:

```
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  }
```

관찰된 동작과 피드백을 바탕으로 도구 정의를 반복적으로 개선함으로써, AI 기반 애플리케이션의 성능과 효율성을 점진적으로 향상시킬 수 있습니다. 도구 정의를 명확하고 간결하게 유지하며, 특정 작업에 초점을 맞추는 것이 중요합니다. 도구 상호작용을 정기적으로 테스트하고 검증하여 원하는 결과와 일치하는지 확인하세요.

도구 구성 및 연결

지금까지 간략히 언급된 도구 사용의 가장 강력한 측면 중 하나는 복잡한 작업을 수행하기 위해 여러 도구를 구성하고 연결할 수 있다는 점입니다. 도구 정의와 입출력 형식을 신중하게 설계함으로써, 다양한 방식으로 조합할 수 있는 재사용 가능한 구성 요소를 만들 수 있습니다.

AI 기반 애플리케이션을 위한 데이터 분석 파이프라인을 구축하는 예시를 살펴보겠습니다. 다음과 같은 도구들이 있을 수 있습니다:

1. **DataRetrieval**: 지정된 기준에 따라 데이터베이스나 API에서 데이터를 가져오는 도구.
2. **DataProcessing**: 검색된 데이터에 대해 계산, 변환 또는 집계를 수행하는 도구.
3. **DataVisualization**: 처리된 데이터를 차트나 그래프와 같은 사용자 친화적인 형식으로 표현하는 도구.

이러한 도구들을 연결함으로써, 관련 데이터를 검색하고, 처리하며, 의미 있는 방식으로 결과를 표현하는 강력한 워크플로우를 만들 수 있습니다. 도구 사용 워크플로우는 다음과 같이 진행될 수 있습니다:

1. LLM이 특정 제품 카테고리의 판매 데이터에 대한 인사이트를 요청하는 사용자 쿼리를 받습니다.
2. LLM이 **DataRetrieval** 도구를 선택하고 데이터베이스에서 관련 판매 데이터를 가져오기 위한 적절한 입력 매개변수를 생성합니다.
3. 검색된 데이터는 **DataProcessing** 도구로 “전달”되어 총 수익, 평균 판매 가격, 성장률과 같은 지표를 계산합니다.
4. 처리된 데이터는 **DataVisualization** 도구에서 분석되어 인사이트를 시각적으로 표현하는 매력적인 차트나 그래프를 생성하고, 차트의 URL을 LLM에 반환합니다.
5. 마지막으로, LLM은 마크다운을 사용하여 시각화된 데이터를 포함하고 주요 발견 사항을 요약한 형식화된 응답을 사용자 쿼리에 대해 생성합니다.

이러한 도구들을 구성함으로써, 애플리케이션에 쉽게 통합할 수 있는 원활한 데이터 분석 워크플로우를 만들 수 있습니다. 이 접근 방식의 장점은 각 도구를 독립적으로 개발하고 테스트할 수 있으며, 다양한 문제를 해결하기 위해 다른 방식으로 조합할 수 있다는 것입니다.

도구의 원활한 구성과 연결을 가능하게 하려면 각 도구에 대해 명확한 입력 및 출력 형식을 정의하는 것이 중요합니다.

예를 들어, **DataRetrieval** 도구는 데이터베이스 연결 세부 정보, 테이블 이름, 쿼리 조건과 같은 매개변수를 받아들이고 결과 집합을 구조화된 JSON 객체로 반환할 수 있습니다. 그런 다음 **DataProcessing** 도구는 이 JSON 객체를 입력으로 받아 변환된 JSON 객체를 출력으로 생성할 수 있습니다. 도구 간의 데이터 흐름을 표준화함으로써 호환성과 재사용성을 보장할 수 있습니다.

도구 생태계를 설계할 때는 애플리케이션의 일반적인 사용 사례를 해결하기 위해 다양한 도구를 어떻게 조합할 수 있는지 생각해보세요. 일반적인 워크플로우나 비즈니스 로직을 캡슐화하는 상위 수준의 도구를 만들어 LLM이 효과적으로 선택하고 사용할 수 있도록 하는 것을 고려하세요.

도구 사용의 힘은 그것이 제공하는 유연성과 모듈성에 있다는 것을 기억하세요. 복잡한 작업을 더 작고 재사용 가능한 도구로 분해함으로써, 다양한 과제를 해결할 수 있는 견고하고 적응력 있는 AI 기반 애플리케이션을 만들 수 있습니다.

향후 방향

AI 기반 애플리케이션 개발 분야가 발전함에 따라, 도구 사용 기능이 더욱 발전할 것으로 예상됩니다. 잠재적인 향후 방향은 다음과 같습니다:

1. **다중 단계 도구 사용:** LLM이 만족스러운 응답을 생성하기 위해 도구를 얼마나 많이 사용해야 하는지 결정할 수 있을 것입니다. 이는 중간 결과를 기반으로 여러 라운드의 도구 선택과 실행을 포함할 수 있습니다.
2. **사전 정의된 도구:** AI 플랫폼이 Python 인터프리터, 웹 검색 도구 또는 일반적인 유ти리티 함수와 같이 개발자가 즉시 활용할 수 있는 사전 정의된 도구 세트를 제공할 수 있습니다.
3. **원활한 통합:** 도구 사용이 더욱 보편화됨에 따라, AI 플랫폼과 인기 있는 개발 프레임워크 간의 더 나은 통합이 이루어져 개발자가 애플리케이션에 도구 사용을 더 쉽게 통합할 수 있을 것으로 예상됩니다.

도구 사용은 개발자가 AI 기반 애플리케이션에서 LLM의 잠재력을 최대한 활용할 수 있게 하는 강력한 기술입니다. LLM을 외부 도구 및 리소스와 연결함으로써, 사용자의 요구에 적응하고 가치 있는 인사이트와 행동을 제공할 수 있는 더욱 동적이고 지능적이며 상황을 인식하는 시스템을 만들 수 있습니다.

도구 사용은 막대한 가능성을 제공하지만, 잠재적인 과제와 고려 사항을 인식하는 것이 중요합니다. 한 가지 핵심 측면은 도구 상호작용의 복잡성을 관리하고 전체 시스템의 안정성과 신뢰성을 보장하는 것입니다. 도구 호출이 실패하거나, 예상치 못한 결과를 반환하거나, 성능에 영향을 미칠 수 있는 시나리오를 처리해야 합니다. 또한 도구의 무단 또는 악의적인 사용을 방지하기 위한 보안 및 접근 제어 조치를 고려해야 합니다. AI 기반 애플리케이션의 무결성과 성능을 유지하기 위해서는 적절한 오류 처리, 로깅 및 모니터링 메커니즘이 중요합니다.

여러분의 프로젝트에서 도구 사용을 탐구할 때, 명확한 목표로 시작하고, 잘 구조화된 도구 정의를 설계하며, 피드백과 결과를 바탕으로 반복하는 것을 잊지 마세요. 올바른 접근 방식과 사고방식을 갖추면, 도구 사용을 통해 AI 기반 애플리케이션에서 새로운 수준의 혁신과 가치를 창출할 수 있습니다

스트림 처리



HTTP를 통한 스트리밍 데이터, 즉 서버 전송 이벤트(SSE)는 클라이언트가 명시적으로 요청하지 않아도 서버가 데이터를 사용 가능한 대로 지속적으로 클라이언트에 전송하는 메커니즘입니다. AI의 응답이 점진적으로 생성되므로, AI의 출력을 생성되는 대로 표시하여 반응성 있는 사용자 경험을 제공하는 것이 합리적입니다. 실제로 제가 알고 있는 모든 AI 제공업체의 API는 완료 엔드포인트에서 스트리밍 응답을 옵션으로 제공하고 있습니다.

이 장이 책에서 [도구 사용하기](#) 바로 다음에 등장하는 이유는 도구 사용과 실시간 AI 응답을 결합했을 때의 강력한 효과 때문입니다. 이를 통해 AI가 사용자 입력을 처리하고, 자유롭게 다양한 도구와 함수를 활용하며, 실시간으로 응답을 제공하는 동적이고 상호작용적인 경험이 가능해집니다.

이러한 원활한 상호작용을 달성하기 위해서는 AI가 호출한 도구 함수 호출과 일반 텍스트 출력을 최종 사용자에게 전달할 수 있는 스트림 핸들러를 작성해야

합니다. 도구 함수 처리 후 반복해야 하는 필요성이 이 작업에 흥미로운 과제를 더합니다.

ReplyStream 구현하기

스트림 처리가 어떻게 구현될 수 있는지 보여주기 위해, 이 장에서는 Olympia에서 사용되는 `ReplyStream` 클래스의 단순화된 버전을 자세히 살펴보겠습니다. 이 클래스의 인스턴스들은 `ruby-openai`와 `openrouter` 같은 AI 클라이언트 라이브러리에서 `stream` 매개변수로 전달될 수 있습니다.

다음은 제가 Olympia의 `PromptSubscriber`에서 `ReplyStream`을 사용하는 방법입니다. 이는 Wisper를 통해 새로운 사용자 메시지 생성을 수신합니다.

```
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...
```

`ReplyStream` 클래스는 이를 인스턴스화한 프롬프트 구독자에 대한 `context` 참조뿐만 아니라, 수신된 데이터의 버퍼를 저장하고 스트림 처리 중에 호출된

함수 이름과 인자들을 추적하기 위한 배열들을 인스턴스 변수로 가지고 있습니다.

```
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
```

`initialize` 메서드는 `ReplyStream` 인스턴스의 초기 상태를 설정하며, 버퍼, 컨텍스트 및 기타 변수들을 초기화합니다.

`call` 메서드는 스트리밍 데이터를 처리하는 주요 진입점입니다. 이 메서드는 (해시로 표현되는) `chunk` 데이터와 선택적 매개변수인 `bytesize`를 받으며, 우리의 예제에서는 `bytesize`를 사용하지 않습니다. 이 메서드 내에서, 클래스는 패턴 매칭을 사용하여 수신된 청크의 구조에 따라 다양한 시나리오를 처리합니다.



청크에 대해 `deep_symbolize_keys`를 호출하면 문자열 대신 심볼을 사용할 수 있게 되어 패턴 매칭을 더 우아하게 작성할 수 있습니다.

```
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
```

우리가 먼저 매칭하는 패턴은 도구 호출과 그에 연관된 함수명입니다. 이를 감지하면 `f_name` 배열에 저장합니다. 함수명을 인덱스 배열에 저장하는 이유는 모델이 병렬 함수 호출이 가능하여 한 번에 여러 함수를 실행하도록 전송할 수 있기 때문입니다.

병렬 함수 호출은 AI 모델이 여러 함수 호출을 동시에 수행하여 이러한 함수 호출의 효과와 결과를 병렬로 처리할 수 있는 기능입니다. 이는 특히 함수

실행에 시간이 오래 걸리는 경우에 유용하며, API와의 왕복 횟수를 줄여 결과적으로 상당한 양의 토큰 사용량을 절약할 수 있습니다.

다음으로 함수 호출에 해당하는 인수들을 매칭해야 합니다.

```
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
```

함수 이름을 처리했던 것과 유사하게, 인자들을 인덱스 배열에 저장합니다.

다음으로, 서버로부터 한 번에 하나의 토큰씩 전달되어 `new_content` 변수에 할당될 일반적인 사용자 대상 메시지를 살펴봅니다. 또한 `finish_reason`도 주시해야 합니다. 이는 출력 시퀀스의 마지막 청크가 될 때까지 `nil` 상태를 유지할 것입니다.

```

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

```

중요한 점은 AI 모델 제공자가 보내는 오류 메시지를 처리하기 위한 패턴 매칭 표현식을 추가한다는 것입니다. 로컬 개발 환경에서는 예외를 발생시키지만, 프로덕션 환경에서는 오류를 로그에 기록하고 종료합니다.

```

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

```

case의 마지막 else 절은 이전의 어떤 패턴도 일치하지 않았을 때 실행됩니다. 이는 단순히 AI 모델이 인식할 수 없는 청크를 보내기 시작할 경우 이를 발견할 수 있도록 하는 안전장치입니다.

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

`send_to_client` 메서드는 버퍼링된 콘텐츠를 클라이언트에게 전송하는 역할을 담당합니다. 이 메서드는 버퍼가 비어있지 않은지 확인하고, 봇 메시지 콘텐츠를 업데이트하며, 봇 메시지를 렌더링하고, 데이터 영속성을 보장하기 위해 콘텐츠를 데이터베이스에 저장합니다.

```
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
```

`finalize` 메서드는 스트림 처리가 완료되었을 때 호출됩니다. 이 메서드는 스트림 중에 수신된 함수 호출이 있는 경우 이를 처리하고, 최종 내용과 기타 관련 정보로 봇 메시지를 업데이트하며, 함수 호출 기록을 초기화합니다.

```
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
```

모델이 함수를 호출하기로 결정하면, 해당 함수 호출(이름과 인자)을 “디스패치”하여 실행되도록 하고 `function_call`과 `function_result` 메시지가 대화 기록에 추가되도록 해야 합니다.

제 경험상, 도구 구현에 의존하는 대신 코드베이스의 한 곳에서 함수 메시지 생성을 처리하는 것이 더 좋습니다. 이는 더 깔끔할 뿐만 아니라 매우 중요한 실용적인 이유도 있습니다. AI 모델이 함수를 호출했는데 루프에서 결과적인 호출과 결과 메시지가 기록에 보이지 않으면, 동일한 함수를 다시 호출할 것입니다. 이는 잠재적으로 영원히 반복될 수 있습니다. AI는 완전히 무상태라는 것을 기억하세요. 따라서 이러한 함수 호출을 다시 AI에게 전달하지 않으면, 그 호출은 발생하지 않은 것과 같습니다.

```

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

```



함수를 호출한 후 기록에 호출 내용과 결과가 남도록 하는 것만큼이나 함수 호출 기록을 지우는 것도 중요합니다. 그래야 루프를 돌 때마다 같은 함수를 반복해서 호출하지 않을 수 있습니다.

“대화 루프”

`PromptSubscriber` 클래스에서는 `PromptDeclarations` 모듈의 `prompt` 메서드를 사용하여 대화 루프의 동작을 정의합니다. `until` 매개변수는 `-> { bot_message.complete? }`로 설정되어 있어, `bot_message`가 완료된 것으로 표시될 때까지 루프가 계속됩니다.

```

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

```



그런데 `bot_message`는 언제 완료된 것으로 표시될까요? 기억이 나지 않는다면 `finalize` 메서드의 13번째 줄을 다시 확인해보세요.

전체 스트림 처리 로직을 검토해보겠습니다.

1. **PromptSubscriber**는 Whisper 발행/구독 시스템을 통해 최종 사용자가 새로운 프롬프트를 생성할 때마다 호출되는 `message_created` 메서드를 통해 새로운 사용자 메시지를 수신합니다.
2. `prompt` 클래스 메서드는 **PromptSubscriber**의 채팅 완성 로직 동작을 선언적으로 정의합니다. AI 모델은 사용자의 메시지 내용, 스트림 매개변수로서의 새로운 `ReplyStream` 인스턴스, 그리고 지정된 루프 조건으로 채팅 완성을 실행합니다.
3. AI 모델이 프롬프트를 처리하고 응답 생성을 시작합니다. 응답이 스트리밍될 때 각 데이터 청크마다 `ReplyStream` 인스턴스의 `call` 메서드가 호출됩니다.
4. AI 모델이 도구 함수를 호출하기로 결정하면, 함수 이름과 인수가 청크에서 추출되어 각각 `f_name`과 `f_arguments` 배열에 저장됩니다.
5. AI 모델이 사용자에게 보여질 내용을 생성하면, 이는 버퍼링되어 `send_to_client` 메서드를 통해 클라이언트로 전송됩니다.
6. 스트림 처리가 완료되면 `finalize` 메서드가 호출됩니다. 스트림 중에 도구 함수가 호출되었다면, 이들은 **PromptSubscriber**의 `dispatch` 메서드를 사용하여 처리됩니다.
7. `dispatch` 메서드는 대화 기록에 `function_call` 메시지를 추가하고, 해당하는 도구 함수를 실행한 다음, 함수 호출 결과와 함께 `function_result` 메시지를 기록에 추가합니다.
8. 도구 함수 처리 후, 이후 루프에서 함수가 중복 호출되는 것을 방지하기 위해 함수 호출 기록이 초기화됩니다.
9. 도구 함수가 호출되지 않았다면, `finalize` 메서드는 `bot_message`를 최종 내용으로 업데이트하고, 완료된 것으로 표시한 후, 업데이트된 메시지를 클라이언트로 전송합니다.
10. 루프 조건 -> { `bot_message.complete?` }가 평가됩니다. `bot_message`가 완료된 것으로 표시되지 않았다면, 루프는 계속되고 업데이트된 대화 기록과 함께 원래 프롬프트가 다시 제출됩니다.

11. `bot_message`가 완료된 것으로 표시될 때까지, 즉 AI 모델이 응답 생성을 마치고 더 이상 실행할 도구 함수가 없다는 것이 확인될 때까지 3-10단계 가 반복됩니다.

이 대화 루프를 구현함으로써, AI 모델이 애플리케이션과 양방향으로 상호작용하며, 필요에 따라 도구 함수를 실행하고 대화가 자연스럽게 종료될 때까지 사용자에게 보여질 응답을 생성할 수 있게 됩니다.

스트림 처리와 대화 루프의 조합을 통해 동적이고 상호작용적인 AI 기반 경험이 가능해지며, AI 모델은 사용자 입력을 처리하고, 다양한 도구와 함수를 활용하며, 진화하는 대화 컨텍스트를 바탕으로 실시간 응답을 제공할 수 있습니다.

자동 연속

AI 출력의 한계를 인식하는 것이 중요합니다. 대부분의 모델은 `max_tokens` 매개변수에 의해 결정되는, 단일 응답에서 생성할 수 있는 최대 토큰 수가 있습니다. AI 모델이 응답을 생성하는 동안 이 한계에 도달하면, 갑자기 중단되고 출력이 잘렸음을 표시합니다.

AI 플랫폼 API의 스트리밍 응답에서, 청크의 `finish_reason` 변수를 검사하여 이러한 상황을 감지할 수 있습니다. `finish_reason`이 "length"로 설정되어 있다면(또는 모델별로 특정한 다른 키 값), 이는 모델이 생성 중에 최대 토큰 한계에 도달했고 출력이 중단되었음을 의미합니다.

이러한 시나리오를 우아하게 처리하고 원활한 사용자 경험을 제공하는 한 가지 방법은 스트림 처리 로직에 자동 연속 메커니즘을 구현하는 것입니다. 길이 관련 종료 이유에 대한 패턴 매칭을 추가함으로써, 중단된 지점에서 자동으로 출력을 계속하도록 루프를 설정할 수 있습니다.

다음은 자동 연속을 지원하기 위해 `ReplyStream` 클래스의 `call` 메서드를 수정하는 의도적으로 단순화된 예시입니다:

```
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
```

이 수정된 버전에서는 `finish_reason`이 잘린 출력을 나타낼 때, 스트림을 종료하는 대신 기록에 메시지 쌍을 추가하되 종료하지 않고, 원래의 사용자 대면 응답 메시지를 `created_at` 속성을 업데이트하여 기록의 “맨 아래”로 이동시킨 다음, AI가 중단된 지점부터 계속 생성할 수 있도록 루프를 실행합니다.

AI 완성 엔드포인트는 상태가 없다는 점을 기억하세요. 기록을 통해 전달하는 내용만 “알고 있습니다”. 이 경우, AI에게 출력이 잘렸다는 것을 알리는 방법은 (최종 사용자에게는 “보이지 않는”) 메시지를 기록에 추가하는 것입니다. 하지만 이것은 의도적으로 단순화된 예시라는 점을 기억하세요. 실제 구현에서는 토큰을 낭비하지 않고 기록에 있는 중복된 어시스턴트 메시지로 AI가 혼란스러워지지 않도록 더 많은 기록 관리가 필요할 것입니다.

자동 연속 생성의 실제 구현에는 무한 루프를 방지하기 위한 이른바 “회로 차단기 로직”이 포함되어야 합니다. 그 이유는 특정 유형의 사용자 프롬프트와 낮은 `max_tokens` 설정이 주어졌을 때, AI가 사용자 대면 출력을 끝없이 계속 루프할 수 있기 때문입니다.

모든 루프마다 별도의 요청이 필요하고, 각 요청마다 전체 기록을 다시 소비한다는 점을 명심하세요. 애플리케이션에 자동 연속 생성을 구현할지 결정할 때는 사용자 경험과 API 사용량 사이의 트레이드오프를 반드시 고려해야 합니다. 특히 프리미엄 상용 모델을 사용할 때는 자동 연속 생성이 위험할 정도로 비용이 많이 들 수 있습니다.

결론

스트림 처리는 도구 사용과 실시간 AI 응답을 결합하는 AI 기반 애플리케이션 구축의 중요한 측면입니다. AI 플랫폼 API의 스트리밍 데이터를 효율적으로 처리함으로써, 원활하고 대화형 사용자 경험을 제공하고, 대용량 응답을 처리하며, 리소스 사용을 최적화하고, 오류를 우아하게 처리할 수 있습니다.

제공된 `Conversation::ReplyStream` 클래스는 패턴 매칭과 이벤트 기반 아키텍처를 사용하여 Ruby 애플리케이션에서 스트림 처리를 구현하는 방법을 보여줍니다. 스트림 처리 기술을 이해하고 활용함으로써, 애플리케이션에서 AI 통합의 잠재력을 최대한 발휘하고 강력하고 매력적인 사용자 경험을 제공할 수 있습니다.

자가 치유 데이터



자가 치유 데이터는 대규모 언어 모델(LLM)의 기능을 활용하여 애플리케이션의 데이터 무결성, 일관성 및 품질을 보장하는 강력한 접근 방식입니다. 이 패턴 카테고리는 AI를 사용하여 데이터 이상, 불일치 또는 오류를 자동으로 감지, 진단 및 수정함으로써 개발자의 부담을 줄이고 높은 수준의 데이터 신뢰성을 유지하는 아이디어에 중점을 둡니다.

자가 치유 데이터 패턴의 핵심은 데이터가 모든 애플리케이션의 생명줄이며, 데이터의 정확성과 무결성을 보장하는 것이 애플리케이션의 적절한 기능과 사용자 경험에 매우 중요하다는 점을 인식하는 것입니다. 하지만 애플리케이션의 규모와 복잡성이 증가함에 따라 데이터 품질을 관리하고 유지하는 것은 복잡하고 시간이 많이 소요되는 작업이 될 수 있습니다. 바로 여기서 AI의 힘이 발휘됩니다.

자가 치유 데이터 패턴에서는 AI 워커들이 애플리케이션의 데이터를 지속적

으로 모니터링하고 분석합니다. 이러한 모델들은 데이터 내의 패턴, 관계 및 이상을 이해하고 해석할 수 있는 능력을 가지고 있습니다. 자연어 처리 및 이해 능력을 활용하여 데이터의 잠재적 문제나 불일치를 식별하고 이를 해결하기 위한 적절한 조치를 취할 수 있습니다.

자가 치유 데이터 프로세스는 일반적으로 다음과 같은 주요 단계를 포함합니다:

1. **데이터 모니터링:** AI 워커들이 애플리케이션의 데이터 스트림, 데이터베이스 또는 저장 시스템을 지속적으로 모니터링하여 이상, 불일치 또는 오류의 징후를 찾습니다. 또는 예외가 발생했을 때 AI 구성 요소를 활성화할 수 있습니다.
2. **이상 감지:** 문제가 감지되면 AI 워커가 데이터를 자세히 분석하여 문제의 구체적인 성격과 범위를 파악합니다. 여기에는 누락된 값, 일관성 없는 형식 또는 미리 정의된 규칙이나 제약 조건을 위반하는 데이터를 감지하는 것이 포함될 수 있습니다.
3. **진단 및 수정:** 문제가 식별되면 AI 워커는 데이터 도메인에 대한 지식과 이해를 바탕으로 적절한 조치 방안을 결정합니다. 여기에는 데이터를 자동으로 수정하거나, 누락된 값을 채우거나, 필요한 경우 사람의 개입을 위해 문제를 표시하는 것이 포함될 수 있습니다.
4. **지속적 학습 (사용 사례에 따라 선택적):** AI 워커가 다양한 데이터 문제를 접하고 해결하면서, 발생한 상황과 대응 방법을 설명하는 출력을 생성할 수 있습니다. 이 메타데이터는 학습 프로세스에 투입되어 시간이 지남에 따라 데이터 이상을 식별하고 해결하는 데 더욱 효과적이고 효율적이 되도록 할 수 있습니다(파인튜닝을 통해 기본 모델도 개선될 수 있음).

데이터 문제를 자동으로 감지하고 수정함으로써 애플리케이션이 고품질의 신뢰할 수 있는 데이터로 운영되도록 보장할 수 있습니다. 이는 애플리케이션의 기능이나 사용자 경험에 영향을 미칠 수 있는 오류, 불일치 또는 데이터 관련 버그의 위험을 줄입니다.

AI 워커가 데이터 모니터링과 수정 작업을 처리하게 되면, 애플리케이션의 다른 중요한 측면에 노력을 집중할 수 있습니다. 이는 수동 데이터 정제와 유지

관리에 소요되는 시간과 리소스를 절약합니다. 실제로 애플리케이션의 규모와 복잡성이 증가함에 따라 데이터 품질을 수동으로 관리하는 것은 점점 더 어려워집니다. “자가 치유 데이터” 패턴은 AI의 힘을 활용하여 대량의 데이터를 처리하고 실시간으로 문제를 감지함으로써 효과적으로 확장됩니다.



AI 모델은 본질적으로 거의 또는 전혀 감독 없이도 시간이 지남에 따라 변화하는 데이터 패턴, 스키마 또는 요구 사항에 적응할 수 있습니다. 의도된 결과에 대한 적절한 지침만 제공된다면, 광범위한 수동 개입이나 코드 변경 없이도 애플리케이션이 새로운 데이터 시나리오를 처리하도록 발전할 수 있습니다.

자가 치유 데이터 패턴은 “다수의 워커”와 같은 우리가 논의한 다른 패턴 카테고리와 잘 부합합니다. 자가 치유 데이터 기능은 데이터 품질과 무결성을 보장하는 데 특별히 중점을 둔 특수한 종류의 워커로 볼 수 있습니다. 이러한 워커는 애플리케이션 기능의 서로 다른 측면에 기여하는 다른 AI 워커들과 함께 작동합니다.

실제로 자가 치유 데이터 패턴을 구현하려면 AI 모델을 애플리케이션 아키텍처에 신중하게 설계하고 통합해야 합니다. 데이터 손실과 손상의 위험 때문에 이 기술을 사용하는 방법에 대한 명확한 지침을 정의해야 합니다. 또한 성능, 확장성 및 데이터 보안과 같은 요소들도 고려해야 합니다.

실제 사례 연구: 손상된 JSON 수정

자가 치유 데이터를 활용하는 가장 실용적이고 편리한 방법 중 하나는 설명하기도 매우 간단합니다: 손상된 JSON을 수정하는 것입니다.

이 기술은 손상된 JSON과 같이 LLM에 의해 생성된 불완전하거나 일관성 없는 데이터를 다루는 일반적인 과제에 적용될 수 있으며, 이러한 문제를 자동으로 감지하고 수정하는 접근 방식을 제공합니다.

Olympia에서 저는 LLM이 완벽하게 유효하지 않은 JSON 데이터를 생성하는 시나리오를 자주 접합니다. 이는 LLM이 실제 JSON 코드 전후에 설명을 추가하거나, 쉼표 누락 또는 이스케이프되지 않은 큰따옴표와 같은 구문 오류를 발생시키는 등 다양한 이유로 발생할 수 있습니다. 이러한 문제는 구문 분석 오류를 일으키고 애플리케이션의 기능에 장애를 초래할 수 있습니다.

이 문제를 해결하기 위해 저는 JsonFixer 클래스 형태로 실용적인 해결책을 구현했습니다. 이 클래스는 손상된 JSON을 입력으로 받아 LLM을 활용하여 가능한 한 많은 정보와 의도를 보존하면서 이를 수정하는 “자가 치유 데이터 (Self-Healing Data)” 패턴을 구현합니다.

```
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
12         transcript << { user: bad_json }
13         transcript << { assistant: "What is the error message?" }
14         transcript << { user: error_message }
15         transcript << { assistant: "Here is the corrected JSON\n\"json\\n\" " }
16
17         self.stop = ["""]
18
19         chat_completion(json: true)
20     end
```

```

20
21 def model
22   "mistralai/mixtral-8x7b-instruct:nitro"
23 end
24 end

```



JsonFixer가 [Ventriloquist](#)를 사용하여 AI의 응답을 안내하는 방식에 주목하세요.

JSON 데이터의 자가 치유 프로세스는 다음과 같이 작동합니다:

1. **JSON 생성:** LLM을 사용하여 특정 프롬프트나 요구사항에 기반한 JSON 데이터를 생성합니다. 하지만 LLM의 특성상, 생성된 JSON이 항상 완벽하게 유효하지는 않을 수 있습니다. 유효하지 않은 JSON을 파서에 입력하면 당연히 `ParserError`가 발생하게 됩니다.

```

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

```

예외 메시지도 `JsonFixer` 호출에 전달되므로, 특히 파서가 데이터의 문제점을 정확히 알려주는 경우가 많기 때문에 데이터의 문제가 무엇인지 완전히 추측할 필요가 없습니다.

2. **LLM 기반 수정:** `JsonFixer` 클래스는 손상된 JSON을 LLM에 다시 전송하며, 원본 정보와 의도를 최대한 보존하면서 JSON을 수정하라는 특정 프롬프트나 지시사항도 함께 전달합니다. 방대한 양의 데이터로 학습되고

JSON 구문을 이해하는 LLM은 오류를 수정하고 유효한 JSON 문자열을 생성하려 시도합니다. [응답 펜싱](#)을 사용하여 LLM의 출력을 제한하며, 이러한 종류의 작업에 특히 적합한 Mixtral 8x7B를 AI 모델로 선택합니다.

3. 겸 중 및 통합: LLM이 반환한 수정된 JSON 문자열은 `chat_completion(json: true)`를 호출했기 때문에 `JSONFixer` 클래스 자체에 의해 파싱됩니다. 수정된 JSON이 유효성 검사를 통과하면 애플리케이션의 워크플로우에 다시 통합되어, 애플리케이션이 데이터 처리를 원활하게 계속할 수 있게 됩니다. 손상된 JSON이 “치유”된 것입니다.

제가 `JSONFixer` 구현체를 여러 번 작성하고 다시 작성했지만, 이 모든 버전들에 투자한 총 시간이 한두 시간을 넘지는 않을 것 같습니다.

자가 치유 데이터 패턴의 핵심 요소는 의도의 보존입니다. LLM 기반 수정 프로세스는 생성된 JSON의 원래 정보와 의도를 최대한 보존하는 것을 목표로 합니다. 이를 통해 수정된 JSON이 의미적 의미를 유지하고 애플리케이션의 맥락에서 효과적으로 사용될 수 있도록 보장합니다.

Olympia에서 이 “자가 치유 데이터” 접근 방식의 실제 구현은 AI, 특히 LLM이 실제 데이터 문제를 해결하는 데 어떻게 활용될 수 있는지를 명확히 보여줍니다. 이는 강력하고 효율적인 애플리케이션을 구축하기 위해 전통적인 프로그래밍 기법과 AI 기능을 결합하는 힘을 보여줍니다.

포스텔의 법칙과 “자가 치유 데이터” 패턴

`JSONFixer` 클래스로 예시되는 “자가 치유 데이터”는 포스텔의 법칙 또는 견고성 원칙으로 알려진 원리와 잘 부합합니다. 포스텔의 법칙은 다음과 같이 명시합니다:

“당신이 하는 일에는 보수적이되, 다른 사람으로부터 받아들이는 것에는

관대하라.”

초기 인터넷의 선구자인 존 포스텔이 처음 표현한 이 원칙은 출력을 보낼 때는 지정된 프로토콜을 엄격히 준수하면서도, 다양하거나 약간 잘못된 입력에도 관대한 시스템을 구축하는 것의 중요성을 강조합니다.

“자가 치유 데이터”의 맥락에서, JSONFixer 클래스는 LLM이 생성한 손상되거나 불완전한 JSON 데이터를 관대하게 수용함으로써 포스텔의 법칙을 구현합니다. 예상된 형식을 엄격히 따르지 않는 JSON을 만났을 때 즉시 거부하거나 실패하지 않습니다. 대신, LLM의 능력을 활용하여 JSON을 수정하려는 관대한 접근 방식을 취합니다.

불완전한 JSON을 관대하게 수용함으로써, JSONFixer 클래스는 견고성과 유연성을 보여줍니다. 실제 세계의 데이터가 다양한 형태로 오며 항상 엄격한 사양을 따르지는 않는다는 것을 인정합니다. 이러한 편차를 우아하게 처리하고 수정함으로써, 클래스는 불완전한 데이터가 있더라도 애플리케이션이 원활하게 계속 작동할 수 있도록 보장합니다.

반면, JSONFixer 클래스는 출력과 관련해서는 포스텔의 법칙의 보수적인 측면도 준수합니다. LLM을 사용하여 JSON을 수정한 후, 클래스는 수정된 JSON이 예상된 형식을 엄격히 준수하는지 확인하기 위해 유효성을 검사합니다. 애플리케이션의 다른 부분으로 전달하기 전에 데이터의 무결성과 정확성을 유지합니다. 이러한 보수적인 접근 방식은 JSONFixer 클래스의 출력이 신뢰할 수 있고 일관성 있도록 보장하여, 상호 운용성을 촉진하고 오류의 전파를 방지합니다.

존 포스텔에 대한 흥미로운 사실:

- 존 포스텔(1943–1998)은 인터넷 발전에 중요한 역할을 한 미국의 컴퓨터 과학자였습니다. 그는 기반 프로토콜과 표준에 대한 중요한 공헌으로 “인터넷의 신”으로 알려졌습니다.
- 포스텔은 인터넷에 관한 기술적, 조직적 노트 시리즈인 RFC(Request for Comments) 문서 시리즈의 편집자였습니다. 그는 TCP, IP, SMTP

와 같은 기초적인 프로토콜을 포함하여 200개 이상의 RFC를 저술하거나 공동 저술했습니다.

- 기술적 공헌 외에도, 포스텔은 겸손하고 협력적인 접근 방식으로 유명했습니다. 그는 견고하고 상호 운용 가능한 네트워크를 구축하기 위해 합의를 도출하고 함께 일하는 것의 중요성을 믿었습니다.
- 포스텔은 1977년부터 1998년 그의 때 이론 죽음까지 USC(University of Southern California)의 정보과학연구소(ISI)에서 컴퓨터 네트워크 부문 디렉터로 재직했습니다.
- 그의 엄청난 공헌을 인정받아, 포스텔은 1998년에 “컴퓨팅계의 노벨상”으로 불리는 권위 있는 튜링상을 사후에 수상했습니다.

JSONFixer 클래스는 Postel이 그의 경력 전반에 걸쳐 중요시했던 핵심 가치인 견고성, 유연성, 상호운용성을 증진시킵니다. 프로토콜을 엄격히 준수하면서도 불완전성에 대해 관대한 시스템을 구축함으로써, 실제 환경의 도전 과제들에 더욱 탄력적이고 적응력 있는 애플리케이션을 만들 수 있습니다.

고려사항 및 금기사항

자가 치유 데이터 접근방식의 적용 가능성은 전적으로 애플리케이션이 다른 데이터의 종류에 따라 달라집니다. 애플리케이션의 모든 JSON 파싱 오류를 자동으로 수정하도록 `JSON.parse`를 단순히 몽키패치하고 싶지 않을 만한 이유가 있습니다. 모든 오류가 자동으로 수정될 수 있거나 수정되어야 하는 것은 아니기 때문입니다.

자가 치유는 특히 데이터 처리 및 가공과 관련된 규제나 준수 요구사항과 결합될 때 매우 까다로워집니다. 의료 및 금융과 같은 일부 산업은 데이터 무결성과 감사 가능성에 대한 규제가 매우 엄격하여, 적절한 감독이나 로깅 없이 “블랙박스” 형태의 데이터 수정을 수행하는 것은 이러한 규제를 위반할 수 있습니다. 여러분이 고안한 자가 치유 데이터 기술이 적용 가능한 법적 및 규제 프레임워

크와 일치하도록 보장하는 것이 매우 중요합니다.

특히 AI 모델을 포함하는 자가 치유 데이터 기술을 적용하는 것은 애플리케이션 성능과 리소스 활용에 큰 영향을 미칠 수 있습니다. 오류 감지와 수정을 위해 AI 모델을 통해 대량의 데이터를 처리하는 것은 계산 집약적일 수 있습니다. 자가 치유 데이터의 이점과 관련된 성능 및 리소스 비용 사이의 균형을 평가하는 것이 중요합니다.

그렇다면 이제 이 강력한 접근방식을 언제 어디에 적용할지 결정하는 데 관련된 요소들을 자세히 살펴보겠습니다.

데이터 중요도

자가 치유 데이터 기술의 적용을 고려할 때, 처리되는 데이터의 중요도를 평가하는 것이 매우 중요합니다. 중요도 수준은 애플리케이션과 비즈니스 도메인의 맥락에서 데이터의 중요성과 민감성을 의미합니다.

경우에 따라, 특히 데이터가 매우 민감하거나 법적 영향을 미치는 경우에는 데이터 오류를 자동으로 수정하는 것이 적절하지 않을 수 있습니다. 다음과 같은 시나리오를 고려해보세요:

- 금융 거래:** 은행 시스템이나 거래 플랫폼과 같은 금융 애플리케이션에서는 데이터 정확성이 가장 중요합니다. 금융 데이터의 사소한 오류조차도 잘못된 계좌 잔액, 잘못 라우팅된 자금, 또는 잘못된 거래 결정과 같은 심각한 결과를 초래할 수 있습니다. 이러한 경우, 철저한 검증과 감사 없이 자동화된 수정을 하는 것은 받아들일 수 없는 위험을 초래할 수 있습니다.
- 의료 기록:** 의료 애플리케이션은 매우 민감하고 기밀인 환자 데이터를 다룹니다. 의료 기록의 부정확성은 환자 안전과 치료 결정에 심각한 영향을 미칠 수 있습니다. 자격을 갖춘 의료 전문가의 적절한 감독과 검증 없이 의료 데이터를 자동으로 수정하는 것은 규제 요구사항을 위반하고 환자의 웰빙을 위험에 빠뜨릴 수 있습니다.

3. **법률 문서:** 계약서, 합의서, 또는 법원 제출 문서와 같은 법률 문서를 다루는 애플리케이션은 엄격한 정확성과 무결성이 요구됩니다. 법률 데이터의 사소한 오류조차도 중대한 법적 영향을 미칠 수 있습니다. 이 영역에서의 자동화된 수정은 적절하지 않을 수 있습니다. 데이터의 유효성과 집행 가능성을 보장하기 위해 법률 전문가의 수동 검토와 검증이 필요한 경우가 많기 때문입니다.

이러한 중요한 데이터 시나리오에서는 자동화된 수정과 관련된 위험이 잠재적 이점보다 큰 경우가 많습니다. 오류를 도입하거나 데이터를 잘못 수정하는 것의 결과는 재정적 손실, 법적 책임, 심지어 개인에 대한 피해로 이어질 수 있습니다.

매우 중요한 데이터를 다룰 때는 수동 검증 및 확인 프로세스를 우선시하는 것이 필수적입니다. 데이터의 정확성과 무결성을 보장하는 데 있어 인간의 감독과 전문성이 매우 중요합니다. 자동화된 자가 치유 기술은 잠재적 오류나 불일치를 표시하는 데 여전히 사용될 수 있지만, 수정에 대한 최종 결정은 인간의 판단과 승인을 포함해야 합니다.

하지만 애플리케이션의 모든 데이터가 동일한 수준의 중요도를 가지고 있지는 않다는 점을 주목해야 합니다. 동일한 애플리케이션 내에서도 덜 민감하거나 오류가 발생해도 영향이 적은 데이터 하위 집합이 있을 수 있습니다. 이러한 경우, 자가 치유 데이터 기술을 해당 특정 데이터 하위 집합에 선택적으로 적용할 수 있으며, 중요 데이터는 수동 검증 대상으로 남겨둘 수 있습니다.

핵심은 애플리케이션의 각 데이터 카테고리의 중요도를 신중하게 평가하고, 관련된 위험과 영향을 기반으로 수정 처리에 대한 명확한 지침과 프로세스를 정의하는 것입니다. 중요한 데이터(예: 원장, 의료 기록)와 중요하지 않은 데이터(예: 우편 주소, 리소스 경고)를 구분함으로써, 적절한 곳에서는 자가 치유 데이터 기술의 이점을 활용하고 필요한 곳에서는 엄격한 통제와 감독을 유지하는 균형을 맞출 수 있습니다.

궁극적으로, 중요 데이터에 자가 치유 데이터 기술을 적용하는 결정은 도메인 전문가, 법률 고문 및 기타 관련 이해관계자와의 협의를 통해 이루어져야 합니다.

다. 애플리케이션의 데이터와 관련된 특정 요구사항, 규제 및 위험을 고려하고 이에 따라 데이터 수정 전략을 조정하는 것이 필수적입니다.

오류 심각도

자가 치유 데이터 기술을 적용할 때, 데이터 오류의 심각도와 영향을 평가하는 것이 중요합니다. 모든 오류가 동등한 것은 아니며, 문제의 심각도에 따라 적절한 조치가 달라질 수 있습니다.

사소한 불일치나 형식 문제는 자동 수정에 적합할 수 있습니다. 예를 들어, 깨진 JSON을 수정하는 자가 치유 데이터 워커는 데이터의 의미나 구조를 크게 변경하지 않고도 누락된 쉼표나 이스케이프되지 않은 큰따옴표를 처리할 수 있습니다. 이러한 유형의 오류는 대개 수정하기 간단하며 전체적인 데이터 무결성에 미치는 영향이 미미합니다.

하지만 데이터의 의미나 무결성을 근본적으로 변화시키는 더 심각한 오류의 경우에는 다른 접근 방식이 필요할 수 있습니다. 이러한 경우에는 자동 수정만으로는 충분하지 않을 수 있으며, 데이터의 정확성과 유효성을 보장하기 위해 인적 개입이 필요할 수 있습니다.

이는 오류의 심각도를 판단하는 데 AI 자체를 활용하는 개념이 등장하는 지점입니다. AI 모델의 기능을 활용함으로써, 단순히 오류를 수정하는 것을 넘어 오류의 심각도를 평가하고 이를 처리하는 방법에 대해 정보에 기반한 결정을 내릴 수 있는 자가 치유 데이터 워커를 설계할 수 있습니다.

예를 들어, 고객 데이터베이스로 유입되는 데이터의 불일치를 수정하는 자가 치유 데이터 워커를 생각해 보겠습니다. 이 워커는 데이터를 분석하고 누락되거나 상충되는 정보와 같은 잠재적 오류를 식별하도록 설계될 수 있습니다. 하지만 모든 오류를 자동으로 수정하는 대신, 워커는 심각한 오류를 사람이 검토할 수 있도록 표시하는 추가적인 도구 호출 기능을 갖출 수 있습니다.

다음은 이를 구현하는 방법의 예시입니다:

```
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"json\n" }
27
28    self.stop = ["""]
29
```

```
30     chat_completion(json: true).then do |result|
31       return if result.blank?
32
33       customer.update(result)
34   end
35 end
36 end
```

이 예시에서 `CustomerDataHealer` 워커는 고객 데이터의 불일치를 식별하고 수정하도록 설계되었습니다. 다시 한 번, 구조화된 출력을 얻기 위해 `Response Fencing` 및 `Ventriloquist`를 사용합니다. 중요한 점은 워커의 시스템 지시사항에 심각한 오류가 발생할 경우 `flag_for_review` 함수를 사용하라는 지침이 포함되어 있다는 것입니다.

워커가 고객 데이터를 처리할 때, 데이터를 분석하고 불일치를 수정하려 시도 합니다. 워커가 오류가 심각하여 사람의 개입이 필요하다고 판단하면, `flag_for_review` 도구를 사용하여 데이터에 플래그를 지정하고 플래그 지정 이유를 제공할 수 있습니다.

`chat_completion` 메서드는 수정된 고객 데이터를 JSON으로 파싱하기 위해 `json: true`와 함께 호출됩니다. 함수 호출 후 반복 처리에 대한 규정이 없으므로, `flag_for_review`가 호출된 경우 결과는 비어있게 됩니다. 그렇지 않은 경우, 고객 데이터는 검토되고 잠재적으로 수정된 데이터로 업데이트됩니다.

오류 심각도 평가와 사람의 검토를 위한 데이터 플래그 지정 옵션을 통합함으로써, 자가 치유 데이터 워커는 더욱 지능적이고 적응력 있게 됩니다. 경미한 오류는 자동으로 처리하면서 심각한 오류는 수동 개입을 위해 전문가에게 에스컬레이션할 수 있습니다.

오류 심각도를 결정하는 구체적인 기준은 도메인 지식과 비즈니스 요구사항을 기반으로 워커의 지시사항에서 정의될 수 있습니다. 데이터 무결성에 대한 영향, 데이터 손실이나 손상 가능성, 부정확한 데이터로 인한 결과 등의 요소들을 심각도 평가 시 고려할 수 있습니다.

AI를 활용하여 오류 심각도를 평가하고 사람의 개입 옵션을 제공함으로써, 자가 치유 데이터 기술은 자동화와 데이터 정확성 유지 사이의 균형을 맞출 수 있습니다. 이러한 접근 방식은 경미한 오류를 효율적으로 수정하면서 심각한 오류는 사람 검토자로부터 필요한 주의와 전문성을 받도록 보장합니다.

도메인 복잡성

자가 치유 데이터 기술의 적용을 고려할 때, 데이터 도메인의 복잡성과 그 구조 및 관계를 지배하는 규칙을 평가하는 것이 중요합니다. 도메인의 복잡성은 자동화된 데이터 수정 접근 방식의 효과성과 실현 가능성에 상당한 영향을 미칠 수 있습니다.

자가 치유 데이터 기술은 데이터가 잘 정의된 패턴과 제약조건을 따를 때 잘 작동합니다. 데이터 구조가 비교적 단순하고 데이터 요소 간의 관계가 명확한 도메인에서는 자동화된 수정을 높은 신뢰도로 적용할 수 있습니다. 예를 들어, 형식 문제를 수정하거나 기본 데이터 유형 제약조건을 적용하는 것은 자가 치유 데이터 워커가 효과적으로 처리할 수 있습니다.

하지만 데이터 도메인의 복잡성이 증가함에 따라 자동화된 데이터 수정과 관련된 과제도 커집니다. 복잡한 비즈니스 로직, 데이터 엔터티 간의 복잡한 관계, 또는 도메인 특화 규칙과 예외가 있는 도메인에서는 자가 치유 데이터 기술이 항상 미묘한 차이를 포착하지 못하고 의도하지 않은 결과를 초래할 수 있습니다.

복잡한 도메인의 예로 금융 거래 시스템을 살펴보겠습니다. 이 도메인에서 데이터는 다양한 금융 상품, 시장 데이터, 거래 규칙 및 규제 요구사항을 포함합니다. 서로 다른 데이터 요소 간의 관계가 복잡할 수 있으며, 데이터 유효성과 일관성을 관리하는 규칙은 도메인에 매우 특화되어 있을 수 있습니다.

이러한 복잡한 도메인에서 거래 데이터의 불일치를 수정하는 자가 치유 데이터 워커는 도메인 특화 규칙과 제약조건에 대한 깊은 이해가 필요합니다. 시장 규제, 거래 한도, 리스크 계산, 결제 절차와 같은 요소들을 고려해야 합니다. 이러한 맥락에서 자동화된 수정은 도메인의 전체 복잡성을 항상 포착하지 못할

수 있으며, 의도치 않게 오류를 도입하거나 도메인 특화 규칙을 위반할 수 있습니다.

도메인 복잡성의 과제를 해결하기 위해, 자가 치유 데이터 기술은 다음과 같은 기술을 통해 도메인 특화 지식과 규칙을 AI 모델과 워커에 통합하여 향상될 수 있습니다:

- 도메인 특화 훈련:** 자가 치유 데이터에 사용되는 AI 모델은 특정 도메인의 복잡성과 규칙을 포착하는 도메인 특화 데이터셋으로 지시되거나 심지어 미세 조정될 수 있습니다. 모델을 대표적인 데이터와 시나리오에 노출시킴으로써, 도메인 특화 패턴, 제약조건 및 예외를 학습할 수 있습니다.
- 규칙 기반 제약조건:** 자가 치유 데이터 워커는 도메인 특화 지식을 인코딩하는 명시적인 규칙 기반 제약조건으로 보강될 수 있습니다. 이러한 규칙은 도메인 전문가에 의해 정의되고 데이터 수정 프로세스에 통합될 수 있습니다. 그러면 AI 모델은 이러한 규칙을 사용하여 결정을 안내하고 도메인 특화 요구사항 준수를 보장할 수 있습니다.
- 도메인 전문가와의 협력:** 복잡한 도메인에서는 자가 치유 데이터 기술의 설계와 개발에 도메인 전문가를 참여시키는 것이 중요합니다. 도메인 전문가는 데이터의 복잡성, 비즈니스 규칙 및 잠재적인 엣지 케이스에 대한 귀중한 통찰력을 제공할 수 있습니다. 그들의 지식은 *Human In The Loop* 패턴을 사용하여 자동화된 데이터 수정의 정확성과 신뢰성을 향상시키기 위해 AI 모델과 워커에 통합될 수 있습니다.
- 점진적이고 반복적인 접근:** 복잡한 도메인을 다룰 때는 자가 치유 데이터에 대해 점진적이고 반복적인 접근 방식을 채택하는 것이 종종 유익합니다. 한 번에 전체 도메인에 대한 수정을 자동화하려고 시도하는 대신, 규칙과 제약조건이 잘 이해되는 특정 하위 도메인이나 데이터 카테고리에 집중합니다. 도메인에 대한 이해가 늘어나고 기술이 효과적임이 입증됨에 따라 자가 치유 기술의 범위를 점진적으로 확장합니다.

데이터 도메인의 복잡성을 고려하고 도메인 특화 지식을 자가 치유 데이터 기술에 통합함으로써 자동화와 정확성 사이의 균형을 맞출 수 있습니다. 자가

치유 데이터가 모든 상황에 적용할 수 있는 만능 해결책이 아니며, 각 도메인의 특정 요구사항과 과제에 맞게 접근 방식을 조정해야 한다는 점을 인식하는 것이 중요합니다.

복잡한 도메인에서는 자가 치유 데이터 기술과 인간의 전문성 및 감독을 결합한 하이브리드 방식이 가장 효과적일 수 있습니다. 자동 수정은 일상적이고 명확하게 정의된 사례를 처리할 수 있으며, 복잡한 시나리오나 예외 상황은 인간의 검토와 개입을 위해 표시될 수 있습니다. 이러한 협력적 접근 방식은 복잡한 데이터 도메인에서 필요한 통제와 정확성을 유지하면서 자동화의 이점을 실현할 수 있도록 보장합니다.

설명 가능성과 투명성

설명 가능성은 AI 모델이 내린 결정의 이유를 이해하고 해석할 수 있는 능력을 의미하며, 투명성은 데이터 수정 과정에 대한 명확한 가시성을 제공하는 것을 포함합니다.

많은 상황에서 데이터 수정은 감사 가능하고 정당화할 수 있어야 합니다. 비즈니스 사용자, 감사자, 규제 기관을 포함한 이해관계자들은 특정 데이터 수정이 이루어진 이유와 AI 모델이 그러한 결정에 도달한 방법에 대한 설명을 요구할 수 있습니다. 이는 금융, 의료, 법률 문제와 같이 데이터의 정확성과 무결성이 중요한 영향을 미치는 도메인에서 특히 중요합니다.

설명 가능성과 투명성에 대한 요구를 해결하기 위해, 자가 치유 데이터 기술은 AI 모델의 의사결정 과정에 대한 통찰력을 제공하는 메커니즘을 포함해야 합니다. 이는 다음과 같은 다양한 접근 방식을 통해 달성할 수 있습니다:

1. **사고 과정:** 모델에게 데이터 변경을 적용하기 전에 “소리내어” 생각하도록 요청하면 의사결정 과정을 더 쉽게 이해할 수 있으며 수정 사항에 대한 사람이 읽을 수 있는 설명을 생성할 수 있습니다. 이에 따른 트레이드오프는 설명과 구조화된 데이터 출력을 분리하는 데 있어 약간의 복잡성이 추가된다는 것인데, 이는…

2. 설명 생성: 자가 치유 데이터 작업자는 수행하는 수정 사항에 대해 사람이 읽을 수 있는 설명을 생성하는 기능을 갖출 수 있습니다. 이는 모델이 데이터 자체에 통합된 의사결정 과정을 이해하기 쉬운 설명으로 출력하도록 함으로써 달성할 수 있습니다. 예를 들어, 자가 치유 데이터 작업자는 식별한 특정 데이터 불일치, 적용한 수정 사항, 그리고 해당 수정의 근거를 강조하는 보고서를 생성할 수 있습니다.
3. 특성 중요도: AI 모델은 지시사항의 일부로 데이터 수정 과정에서 서로 다른 특성이나 속성의 중요도에 대한 정보를 받을 수 있습니다. 이러한 지시사항은 다시 인간 이해관계자들에게 공개될 수 있습니다. 모델의 결정에 영향을 미치는 주요 요인들을 식별함으로써, 이해관계자들은 수정 사항의 근거를 이해하고 그 타당성을 평가할 수 있습니다.
4. 로깅과 감사: 자가 치유 데이터 프로세스의 투명성을 유지하기 위해 포괄적인 로깅 및 감사 메커니즘을 구현하는 것이 중요합니다. AI 모델이 수행하는 모든 데이터 수정은 원본 데이터, 수정된 데이터, 그리고 취해진 특정 조치를 포함하여 기록되어야 합니다. 이러한 감사 추적을 통해 소급 분석이 가능하며 데이터에 대한 수정 사항의 명확한 기록을 제공합니다.
5. 인간 참여형 접근 방식: 인간 참여형 접근 방식을 통합하면 자가 치유 데이터 기술의 설명 가능성과 투명성을 향상시킬 수 있습니다. AI가 생성한 수정 사항의 검토와 검증에 전문가를 참여시킴으로써, 조직은 수정 사항이 도메인 지식과 비즈니스 요구사항에 부합하도록 보장할 수 있습니다. 인간의 감독은 추가적인 책임성 계층을 제공하며 AI 모델의 잠재적 편향이나 오류를 식별할 수 있게 합니다.
6. 지속적인 모니터링과 평가: 자가 치유 데이터 기술의 성능을 정기적으로 모니터링하고 평가하는 것은 투명성과 신뢰를 유지하는 데 필수적입니다. 시간이 지남에 따라 AI 모델의 정확성과 효과성을 평가함으로써, 조직은 모든 편차나 이상을 식별하고 시정 조치를 취할 수 있습니다. 지속적인 모니터링은 자가 치유 데이터 프로세스가 신뢰할 수 있고 원하는 결과에 부합하도록 보장하는 데 도움이 됩니다.

설명 가능성과 투명성은 자가 치유 데이터 기술을 구현할 때 중요한 고려사항

입니다. 데이터 수정에 대한 명확한 설명을 제공하고, 포괄적인 감사 추적을 유지하며, 인간의 감독을 포함함으로써, 조직은 자가 치유 데이터 프로세스에 대한 신뢰를 구축하고 데이터에 대한 수정이 정당화되며 비즈니스 목표에 부합하도록 보장할 수 있습니다.

자동화의 이점과 투명성의 필요성 사이에서 균형을 맞추는 것이 중요합니다. 자가 치유 데이터 기술이 데이터 품질과 효율성을 크게 향상시킬 수 있지만, 이는 데이터 수정 프로세스에 대한 가시성과 통제력을 잃는 대가로 이루어져서는 안 됩니다. 설명 가능성과 투명성을 염두에 두고 자가 치유 데이터 작업자를 설계함으로써, 조직은 필요한 수준의 책임성과 데이터에 대한 신뢰를 유지하면서 AI의 힘을 활용할 수 있습니다.

의도하지 않은 결과

자가 치유 데이터 기술이 데이터 품질과 일관성을 개선하는 것을 목표로 하지만, 의도하지 않은 결과가 발생할 수 있다는 점을 인식하는 것이 중요합니다. 신중하게 설계되고 모니터링되지 않는 자동 수정은 의도치 않게 데이터의 의미나 맥락을 변경하여 후속 문제를 초래할 수 있습니다.

자가 치유 데이터의 주요 위험 중 하나는 데이터 수정 과정에서 편향이나 오류가 도입될 수 있다는 점입니다. AI 모델은 다른 소프트웨어 시스템과 마찬가지로 학습 데이터에 존재하는 편향이나 알고리즘 설계를 통해 도입된 편향의 영향을 받을 수 있습니다. 이러한 편향이 식별되고 완화되지 않으면, 자가 치유 데이터 프로세스를 통해 전파되어 왜곡되거나 부정확한 데이터 수정을 초래할 수 있습니다.

예를 들어, 고객 인구통계 데이터의 불일치를 수정하는 자가 치유 데이터 워커를 고려해보겠습니다. AI 모델이 과거 데이터로부터 편향성을 학습했다면, 예를 들어 특정 직업이나 소득 수준을 특정 성별이나 민족과 연관 짓는 경우, 잘못된 가정을 하고 그러한 편향성을 강화하는 방식으로 데이터를 수정할 수 있습니다. 이는 부정확한 고객 프로필, 잘못된 비즈니스 결정, 그리고 잠재적으로 차별적인 결과를 초래할 수 있습니다.

또 다른 잠재적인 의도하지 않은 결과는 데이터 수정 과정에서 가치 있는 정보나 맥락이 손실되는 것입니다. 자가 치유 데이터 기술은 주로 일관성을 보장하기 위해 데이터를 표준화하고 정규화하는 데 중점을 둡니다. 하지만 경우에 따라서는 원본 데이터가 전체 그림을 이해하는 데 중요한 미묘한 차이, 예외 사항 또는 맥락 정보를 포함할 수 있습니다. 맹목적으로 표준화를 강제하는 자동 수정은 이러한 가치 있는 정보를 의도치 않게 제거하거나 모호하게 만들 수 있습니다. 예를 들어, 의료 기록의 불일치를 수정하는 자가 치유 데이터 워커를 상상해보세요. 워커가 희귀 질환이나 특이한 치료 계획이 있는 환자의 의료 기록을 접했을 때, 더 일반적인 패턴에 맞추기 위해 데이터를 정규화하려 할 수 있습니다. 하지만 그렇게 함으로써, 환자의 고유한 상황을 정확하게 표현하는 데 중요한 구체적인 세부 사항과 맥락을 잃을 수 있습니다. 이러한 정보의 손실은 환자 치료와 의료 의사 결정에 심각한 영향을 미칠 수 있습니다.

의도하지 않은 결과의 위험을 줄이기 위해서는 자가 치유 데이터 기술을 설계하고 구현할 때 선제적인 접근 방식을 취하는 것이 필수적입니다:

- 1. 철저한 테스트와 검증:** 자가 치유 데이터 워커를 프로덕션 환경에 배포하기 전에, 다양한 시나리오에 대해 그들의 동작을 철저히 테스트하고 검증하는 것이 중요합니다. 여기에는 다양한 엣지 케이스, 예외 사항 및 잠재적 편향성을 포함하는 대표적인 데이터셋으로 테스트하는 것이 포함됩니다. 엄격한 테스트는 실제 데이터에 영향을 미치기 전에 의도하지 않은 결과를 식별하고 해결하는 데 도움이 됩니다.
- 2. 지속적인 모니터링과 평가:** 시간이 지남에 따라 의도하지 않은 결과를 감지하고 완화하기 위해 지속적인 모니터링과 평가 메커니즘을 구현하는 것이 필수적입니다. 자가 치유 데이터 프로세스의 결과를 정기적으로 검토하고, 다운스트림 시스템과 의사 결정에 미치는 영향을 분석하며, 이해관계자들로부터 피드백을 수집하는 것은 부작용을 식별하고 적시에 시정 조치를 취하는 데 도움이 될 수 있습니다. 조직에 운영 대시보드가 있다면, 자동화된 데이터 변경과 관련된 지표를 명확하게 표시하는 것이 좋은 아이디어일 것입니다. 정상적인 데이터 변경 활동에서 크게 벗어나는

경우에 대한 경보를 추가하는 것은 더 좋은 아이디어일 것입니다!

3. 인간의 감독과 개입: 자가 치유 데이터 프로세스에 대한 인간의 감독과 개입 능력을 유지하는 것이 중요합니다. 자동화가 효율성을 크게 향상시킬 수 있지만, 특히 중요하거나 민감한 영역에서는 AI 모델이 수행한 수정 사항을 인간 전문가가 검토하고 검증하는 것이 중요합니다. 인간의 판단과 도메인 전문성은 발생할 수 있는 의도하지 않은 결과를 식별하고 해결하는데 도움이 될 수 있습니다.
4. 설명 가능한 AI (XAI)와 투명성: 이전 하위 섹션에서 논의한 바와 같이, 설명 가능한 AI 기술을 통합하고 자가 치유 데이터 프로세스의 투명성을 보장하는 것은 의도하지 않은 결과를 완화하는 데 도움이 될 수 있습니다. 데이터 수정에 대한 명확한 설명을 제공하고 포괄적인 감사 추적을 유지함으로써, 조직은 AI 모델이 수행한 수정 사항의 근거를 더 잘 이해하고 추적할 수 있습니다.
5. 점진적이고 반복적인 접근: 자가 치유 데이터에 대한 점진적이고 반복적인 접근 방식을 채택하면 의도하지 않은 결과의 위험을 최소화하는 데 도움이 될 수 있습니다. 전체 데이터셋에 한 번에 자동 수정을 적용하는 대신, 데이터의 일부분부터 시작하여 기술이 효과적이고 신뢰할 수 있다고 입증됨에 따라 점진적으로 범위를 확장하십시오. 이를 통해 과정에서 신중한 모니터링과 조정이 가능하며, 의도하지 않은 결과의 영향을 줄일 수 있습니다.
6. 협력과 피드백: 자가 치유 데이터 프로세스 전반에 걸쳐 다양한 도메인의 이해관계자들을 참여시키고 협력과 피드백을 장려하는 것은 의도하지 않은 결과를 식별하고 해결하는 데 도움이 될 수 있습니다. 도메인 전문가, 데이터 소비자 및 최종 사용자로부터 정기적으로 의견을 구하는 것은 데이터 수정의 실제 영향에 대한 귀중한 통찰력을 제공하고 간과되었을 수 있는 문제를 강조할 수 있습니다.

의도하지 않은 결과의 위험을 선제적으로 해결하고 적절한 안전장치를 구현함으로써, 조직은 잠재적인 부작용을 최소화하면서 자가 치유 데이터 기술의

이점을 활용할 수 있습니다. 자가 치유 데이터를 반복적이고 협력적인 프로세스로 접근하여, 원하는 결과와 일치하고 데이터의 무결성과 신뢰성을 유지하도록 기술을 지속적으로 모니터링, 평가 및 개선하는 것이 중요합니다.

자가 치유 데이터 패턴의 사용을 고려할 때, 이러한 요소들을 신중히 평가하고 잠재적 위험과 한계에 대비하여 이점을 저울질하는 것이 필수적입니다. 경우에 따라서는 자동화된 수정과 인간의 감독 및 개입을 결합한 하이브리드 접근 방식이 가장 적절한 해결책일 수 있습니다.

또한 자가 치유 데이터 기술이 강력한 데이터 검증, 입력 데이터 정제 및 오류 처리 메커니즘을 대체하는 것으로 여겨서는 안 된다는 점에 주목할 가치가 있습니다. 이러한 기본적인 관행들은 데이터 무결성과 보안을 보장하는 데 여전히 중요합니다. 자가 치유 데이터는 이러한 기존 조치들을 보완하고 강화할 수 있는 보완적인 접근 방식으로 보아야 합니다.

궁극적으로, 자가 치유 데이터 패턴을 사용할지 여부는 애플리케이션의 특정 요구사항, 제약 조건 및 우선순위에 달려 있습니다. 위에서 설명한 고려사항들을 신중히 검토하고 이를 애플리케이션의 목표와 아키텍처에 맞춤으로써, 자가 치유 데이터 기술을 언제 어떻게 효과적으로 활용할지에 대해 정보에 입각한 결정을 내릴 수 있습니다.

맥락 기반 콘텐츠 생성



맥락 기반 콘텐츠 생성 패턴은 대규모 언어 모델(LLM)의 능력을 활용하여 애플리케이션 내에서 동적이고 맥락에 특화된 콘텐츠를 생성합니다. 이러한 패턴 카테고리는 사용자의 특정 요구사항, 선호도, 심지어 애플리케이션과의 이전 및 현재 상호작용을 기반으로 개인화되고 관련성 있는 콘텐츠를 제공하는 것의 중요성을 인식합니다.

이러한 접근 방식에서 “콘텐츠”는 주요 콘텐츠(즉, 블로그 게시물, 기사 등)와 주요 콘텐츠에 대한 추천과 같은 메타 콘텐츠를 모두 지칭합니다.

맥락 기반 콘텐츠 생성 패턴은 사용자 참여도를 높이고, 맞춤형 경험을 제공하며, 여러분과 사용자를 위한 콘텐츠 생성 작업을 자동화하는 데 중요한 역할을

할 수 있습니다. 이 장에서 설명하는 패턴들을 활용함으로써, 맥락과 입력에 실시간으로 적응하며 동적으로 콘텐츠를 생성하는 애플리케이션을 만들 수 있습니다.

이 패턴들은 LLM을 사용자 인터페이스(때로는 “크롬”이라고 함)부터 이메일과 기타 알림 형태, 그리고 모든 콘텐츠 생성 파이프라인에 이르기까지 애플리케이션의 출력에 통합하는 방식으로 작동합니다.

사용자가 애플리케이션과 상호작용하거나 특정 콘텐츠를 요청할 때, 애플리케이션은 사용자 선호도, 이전 상호작용 또는 특정 프롬프트와 같은 관련 맥락을 캡처합니다. 이러한 맥락 정보는 필요한 템플릿이나 지침과 함께 LLM에 입력되어, 그렇지 않았다면 하드코딩되거나 데이터베이스에 저장되거나 알고리즘적으로 생성되어야 했을 텍스트 출력을 생성하는 데 사용됩니다.

LLM이 생성하는 콘텐츠는 개인화된 추천, 동적 제품 설명, 맞춤형 이메일 응답, 또는 전체 기사나 블로그 게시물과 같은 다양한 형태를 취할 수 있습니다. 제가 1년 전에 선구적으로 시도한 가장 혁신적인 사용 사례 중 하나는 폼 레이블, 툴팁 및 기타 설명 텍스트와 같은 UI 요소를 동적으로 생성하는 것입니다.

개인화

맥락 기반 콘텐츠 생성 패턴의 주요 이점 중 하나는 사용자에게 고도로 개인화된 경험을 제공할 수 있다는 것입니다. 사용자별 맥락을 기반으로 콘텐츠를 생성함으로써, 이러한 패턴들은 애플리케이션이 개별 사용자의 관심사, 선호도 및 상호작용에 맞춘 콘텐츠를 제공할 수 있게 합니다.

개인화는 단순히 일반적인 콘텐츠에 사용자의 이름을 삽입하는 것 이상을 의미합니다. 각 사용자에 대해 사용 가능한 풍부한 맥락을 활용하여 그들의 특정 요구와 욕구에 부합하는 콘텐츠를 생성하는 것을 포함합니다. 이러한 맥락에는 다음과 같은 다양한 요소들이 포함될 수 있습니다:

1. 사용자 프로필 정보: 이 기술을 적용하는 가장 일반적인 수준에서, 인구

통계 데이터, 관심사, 선호도 및 기타 프로필 속성을 사용하여 사용자의 배경과 특성에 부합하는 콘텐츠를 생성할 수 있습니다.

2. **행동 데이터:** 조회한 페이지, 클릭한 링크 또는 구매한 제품과 같은 애플리케이션과의 과거 상호작용은 사용자의 행동과 관심사에 대한 귀중한 통찰력을 제공할 수 있습니다. 이 데이터는 사용자의 참여 패턴을 반영하고 미래의 요구를 예측하는 콘텐츠 제안을 생성하는 데 사용될 수 있습니다.
3. **맥락적 요소:** 사용자의 현재 위치, 사용 기기, 시간대, 심지어 날씨와 같은 현재 맥락이 콘텐츠 생성 과정에 영향을 미칠 수 있습니다. 예를 들어, 여행 애플리케이션은 AI 워커를 통해 사용자의 현재 위치와 현재 날씨 조건을 기반으로 개인화된 추천을 생성할 수 있습니다.

이러한 맥락적 요소들을 활용함으로써, 맥락 기반 콘텐츠 생성 패턴은 각 개별 사용자에게 맞춤 제작된 것처럼 느껴지는 콘텐츠를 제공할 수 있습니다. 이러한 수준의 개인화는 다음과 같은 여러 가지 중요한 이점을 제공합니다:

1. **향상된 참여도:** 개인화된 콘텐츠는 사용자의 관심을 끌고 애플리케이션과의 지속적인 상호작용을 유도합니다. 사용자가 콘텐츠가 관련성이 있고 자신의 요구에 직접적으로 부응한다고 느낄 때, 애플리케이션과 더 오래 상호작용하고 기능을 탐색할 가능성이 높아집니다.
2. **향상된 사용자 만족도:** 개인화된 콘텐츠는 애플리케이션이 사용자의 고유한 요구사항을 이해하고 중요하게 생각한다는 것을 보여줍니다. 도움이 되고 유익하며 사용자의 관심사에 부합하는 콘텐츠를 제공함으로써, 애플리케이션은 사용자 만족도를 높이고 사용자와 더 강력한 유대감을 형성할 수 있습니다.
3. **높은 전환율:** 전자상거래나 마케팅 애플리케이션의 맥락에서, 개인화된 콘텐츠는 전환율에 상당한 영향을 미칠 수 있습니다. 사용자의 선호도와 행동에 맞춤화된 제품, 제안 또는 추천을 제시함으로써, 애플리케이션은 구매나 서비스 가입과 같은 원하는 행동을 취할 가능성을 높일 수 있습니다.

생산성

맥락 기반 콘텐츠 생성 패턴은 창의적 프로세스에서 수동 콘텐츠 생성과 편집의 필요성을 줄임으로써 특정 유형의 생산성을 크게 향상시킬 수 있습니다. LLM의 능력을 활용함으로써, 콘텐츠 제작자와 개발자들이 지루한 수동 작업에 시간과 노력을 들여야 했을 고품질 콘텐츠를 대규모로 생성할 수 있습니다.

전통적으로 콘텐츠 제작자들은 애플리케이션의 요구사항과 사용자의 기대를 충족시키기 위해 연구, 작성, 편집, 그리고 형식 지정 작업을 수행해야 했습니다. 이러한 과정은 콘텐츠의 양이 늘어날수록 더욱 많은 시간과 자원이 소요되는 작업이었습니다.

하지만 맥락적 콘텐츠 생성 패턴을 활용하면 콘텐츠 제작 과정을 대부분 자동화할 수 있습니다. LLM은 제공된 프롬프트와 지침을 바탕으로 일관성 있고, 문법적으로 정확하며, 맥락에 맞는 콘텐츠를 생성할 수 있습니다. 이러한 자동화는 다음과 같은 생산성 이점을 제공합니다:

1. **수동 작업 감소:** 콘텐츠 생성 작업을 LLM에 위임함으로써, 콘텐츠 제작자들은 콘텐츠 전략, 아이디어 구상, 품질 보증과 같은 더 높은 수준의 작업에 집중할 수 있습니다. 그들은 LLM에 필요한 맥락, 템플릿, 지침을 제공하고 실제 콘텐츠 생성은 LLM이 처리하도록 할 수 있습니다. 이는 작성과 편집에 필요한 수동 작업을 줄여 콘텐츠 제작자의 생산성과 효율성을 높여줍니다.
2. **더 빠른 콘텐츠 생성:** LLM은 인간 작성자보다 훨씬 빠르게 콘텐츠를 생성할 수 있습니다. 적절한 프롬프트와 지침이 있다면, LLM은 몇 초 또는 몇 분 안에 여러 개의 콘텐츠를 생성할 수 있습니다. 이러한 속도는 애플리케이션이 사용자의 요구와 끊임없이 변화하는 디지털 환경에 발맞춰 훨씬 더 빠른 속도로 콘텐츠를 생성할 수 있게 합니다.

더 빠른 콘텐츠 생성이 “공유지의 비극” 상황을 초래하여 아무도 읽지 않는 콘텐츠로 인터넷이 넘쳐나고 있는 것은 아닐까요? 안타깝게도 그렇다고 생각합니다.

3. 일관성과 품질: LLM은 콘텐츠의 스타일, 톤, 품질을 일관되게 수정하는 것이 매우 쉽습니다. 명확한 지침과 예시가 제공된다면, 특정 종류의 애플리케이션(예: 뉴스룸, PR 등)은 인간이 생성한 콘텐츠가 브랜드의 톤앤매너와 일치하고 원하는 품질 기준을 충족하도록 보장할 수 있습니다. 이러한 일관성은 광범위한 편집과 수정의 필요성을 줄여 콘텐츠 제작 과정에서 시간과 노력을 절약합니다.
4. 반복과 최적화: 맥락적 콘텐츠 생성 패턴은 콘텐츠의 빠른 반복과 최적화를 가능하게 합니다. LLM에 제공되는 프롬프트, 템플릿 또는 지침을 조정함으로써, 여러분의 애플리케이션은 과거에는 불가능했던 방식으로 콘텐츠의 변형을 빠르게 생성하고 다양한 접근 방식을 자동으로 테스트할 수 있습니다. 이러한 반복적인 과정을 통해 콘텐츠 전략을 더 빠르게 실험하고 개선할 수 있어, 시간이 지남에 따라 더 효과적이고 매력적인 콘텐츠를 만들 수 있습니다. 이 특정 기술은 이탈률과 참여도를 기반으로 성패가 결정되는 이커머스와 같은 애플리케이션에서 완전한 게임 체인저가 될 수 있습니다.



맥락적 콘텐츠 생성 패턴이 생산성을 크게 향상시킬 수 있지만, 인간의 참여 필요성을 완전히 제거하지는 않는다는 점을 주목해야 합니다. 콘텐츠 제작자와 편집자들은 여전히 전반적인 콘텐츠 전략을 정의하고, LLM에 지침을 제공하며, 생성된 콘텐츠의 품질과 적절성을 확인하는 중요한 역할을 수행합니다.

맥락적 콘텐츠 생성 패턴은 콘텐츠 생성의 더 반복적이고 시간이 많이 소요되는 측면을 자동화함으로써, 더 가치 있는 작업에 투입할 수 있는 귀중한 인적

시간과 자원을 확보할 수 있게 합니다. 이러한 생산성 향상을 통해 콘텐츠 생성 워크플로우를 최적화하면서 사용자에게 더 개인화되고 매력적인 콘텐츠를 제공할 수 있습니다.

빠른 반복과 실험

맥락적 콘텐츠 생성 패턴을 통해 다양한 콘텐츠 변형을 빠르게 반복하고 실험할 수 있어, 콘텐츠 전략을 더 빠르게 최적화하고 개선할 수 있습니다. 모델에 제공되는 맥락, 템플릿 또는 지침을 조정하는 것만으로도 몇 초 만에 여러 버전의 콘텐츠를 생성할 수 있습니다.

이러한 빠른 반복 기능은 다음과 같은 주요 이점을 제공합니다:

- 테스트와 최적화:** 콘텐츠 변형을 빠르게 생성할 수 있는 능력을 통해 다양한 접근 방식을 쉽게 테스트하고 그 효과를 측정할 수 있습니다. 예를 들어, 특정 사용자 세그먼트나 맥락에 맞춘 제품 설명이나 마케팅 메시지의 여러 버전을 생성할 수 있습니다. 클릭률이나 전환율과 같은 사용자 참여 지표를 분석함으로써 가장 효과적인 콘텐츠 변형을 식별하고 그에 따라 콘텐츠 전략을 최적화할 수 있습니다.
- A/B 테스트:** 맥락적 콘텐츠 생성 패턴을 통해 콘텐츠의 원활한 A/B 테스트가 가능합니다. 두 개 이상의 콘텐츠 변형을 생성하여 서로 다른 사용자 그룹에게 무작위로 제공할 수 있습니다. 각 변형의 성과를 비교함으로써 어떤 콘텐츠가 목표 고객층에게 가장 잘 받아들여지는지 파악할 수 있습니다. 이러한 데이터 기반 접근 방식을 통해 정보에 기반한 결정을 내리고 사용자 참여를 최대화하며 원하는 결과를 달성하기 위해 콘텐츠를 지속적으로 개선할 수 있습니다.
- 개인화 실험:** 빠른 반복과 실험은 개인화에 있어 특히 가치가 있습니다. 맥락적 콘텐츠 생성 패턴을 사용하면 다양한 사용자 세그먼트, 선호도 또는 행동을 기반으로 개인화된 콘텐츠 변형을 빠르게 생성할 수 있습니다.

다양한 개인화 전략을 실험함으로써 개별 사용자를 참여시키고 맞춤형 경험을 제공하는 가장 효과적인 접근 방식을 식별할 수 있습니다.

4. 변화하는 트렌드 적응: 빠른 반복과 실험을 할 수 있는 능력은 변화하는 트렌드와 사용자 선호도에 민첩하게 적응할 수 있게 해줍니다. 새로운 주제, 키워드 또는 사용자 행동이 등장할 때, 이러한 트렌드에 맞는 콘텐츠를 신속하게 생성할 수 있습니다. 지속적인 실험과 콘텐츠 개선을 통해 끊임없이 진화하는 디지털 환경에서 경쟁력을 유지하고 관련성을 유지할 수 있습니다.
5. 비용 효율적인 실험: 전통적인 콘텐츠 실험은 콘텐츠 제작자가 수동으로 다양한 변형을 개발하고 테스트해야 하므로 상당한 시간과 자원이 필요합니다. 하지만 맥락 기반 콘텐츠 생성 패턴을 사용하면 실험 비용이 크게 줄어듭니다. LLM은 콘텐츠 변형을 빠르고 대규모로 생성할 수 있어, 큰 비용을 들이지 않고도 다양한 아이디어와 접근 방식을 탐색할 수 있습니다.

빠른 반복과 실험을 최대한 활용하려면 잘 정의된 실험 프레임워크를 갖추는 것이 중요합니다. 이 프레임워크에는 다음이 포함되어야 합니다:

- 각 실험에 대한 명확한 목표와 가설
- 콘텐츠 성과를 측정하기 위한 적절한 지표와 추적 메커니즘
- 관련 콘텐츠 변형이 적절한 사용자에게 제공되도록 하는 세분화 및 타겟팅 전략
- 실험 데이터로부터 인사이트를 도출하기 위한 분석 및 보고 도구
- 학습 내용과 최적화를 콘텐츠 전략에 반영하는 프로세스

빠른 반복과 실험을 수용함으로써 콘텐츠를 지속적으로 개선하고 최적화하여 애플리케이션의 목표를 달성하는 데 효과적이고 관련성 있으며 매력적인 상태를 유지할 수 있습니다. 콘텐츠 제작에 대한 이러한 민첩한 접근 방식을 통해 시장을 선도하고 탁월한 사용자 경험을 제공할 수 있습니다.

확장성과 효율성

애플리케이션이 성장하고 개인화된 콘텐츠에 대한 수요가 증가함에 따라, 맥락 기반 콘텐츠 생성 패턴은 콘텐츠 제작의 효율적인 확장을 가능하게 합니다. LLM은 인적 자원을 비례적으로 늘리지 않고도 많은 수의 사용자와 맥락에 대해 동시에 콘텐츠를 생성할 수 있습니다. 이러한 확장성을 통해 애플리케이션은 콘텐츠 제작 능력에 부담을 주지 않으면서도 증가하는 사용자 기반에 개인화된 경험을 제공할 수 있습니다.



맥락 기반 콘텐츠 생성은 애플리케이션을 “즉시” 국제화하는 데 효과적으로 사용될 수 있습니다. 실제로 제가 Instant18n Gem을 사용하여 Olympia를 1년도 안 되는 기간 동안 6개 이상의 언어로 제공한 것이 바로 그 예입니다.

AI 기반 현지화

잠시 자랑을 하자면, Rails 앱을 위한 제 Instant18n 라이브러리는 “맥락 기반 콘텐츠 생성” 패턴의 혁신적인 예시이며, 애플리케이션 개발에서 AI의 변혁적 잠재력을 보여줍니다. 이 gem은 OpenAI의 GPT 대규모 언어 모델의 힘을 활용하여 Rails 애플리케이션에서 국제화와 현지화를 처리하는 방식을 혁신합니다.

전통적으로 Rails 애플리케이션의 국제화는 번역 키를 수동으로 정의하고 지원되는 각 언어에 대한 해당 번역을 제공하는 과정을 포함합니다. 이 과정은 시간이 많이 소요되고, 자원 집약적이며, 불일치가 발생하기 쉽습니다. 하지만 Instant18n gem을 사용하면 현지화의 패러다임이 완전히 재정의됩니다.

대규모 언어 모델을 통합함으로써 Instant18n gem은 텍스트의 맥락과 의미를 기반으로 실시간 번역을 생성할 수 있게 합니다. 미리 정의된 번역 키와 정적 번역에 의존하는 대신, 이 gem은 AI의 힘을 사용하여 텍스트를 동적으로 번역 합니다. 이러한 접근 방식은 다음과 같은 주요 이점을 제공합니다:

1. 원활한 현지화: Instant18n gem을 사용하면 개발자가 더 이상 지원되는 각 언어에 대한 번역 파일을 수동으로 정의하고 유지할 필요가 없습니다. 이 gem은 제공된 텍스트와 원하는 대상 언어를 기반으로 자동으로 번역을 생성하여 현지화 과정을 수월하고 원활하게 만듭니다.
2. 맥락적 정확성: AI는 번역되는 텍스트의 뉘앙스를 파악하기에 충분한 맥락을 제공받을 수 있습니다. 주변 맥락, 관용구, 문화적 참조를 고려하여 정확하고 자연스러우며 맥락적으로 적절한 번역을 생성할 수 있습니다.
3. 광범위한 언어 지원: Instant18n gem은 GPT의 방대한 지식과 언어 능력을 활용하여 광범위한 언어로의 번역을 가능하게 합니다. 스페인어와 프랑스어 같은 일반적인 언어부터 클링온어와 엘프어 같은 더 희귀하거나 가상의 언어까지, 이 gem은 다양한 번역 요구 사항을 처리할 수 있습니다.
4. 유연성과 창의성: 이 gem은 전통적인 언어 번역을 넘어서 창의적이고 비전통적인 현지화 옵션을 제공합니다. 개발자는 텍스트를 다양한 스타일, 방언, 심지어 가상의 언어로도 번역할 수 있어, 독특한 사용자 경험과 매력적인 콘텐츠를 위한 새로운 가능성을 열어줍니다.
5. 성능 최적화: Instant18n gem은 성능을 향상시키고 반복되는 번역의 오버헤드를 줄이기 위한 캐싱 메커니즘을 포함합니다. 번역된 텍스트는 캐시되어 동일한 번역에 대한 후속 요청이 중복된 API 호출 없이 빠르게 처리될 수 있습니다.

Instant18n gem은 AI를 활용하여 동적으로 현지화된 콘텐츠를 생성함으로써 “맥락 기반 콘텐츠 생성” 패턴의 힘을 보여줍니다. 이는 AI가 Rails 애플리케이션의 핵심 기능에 어떻게 통합되어 개발자들이 국제화와 현지화에 접근하는 방식을 변화시킬 수 있는지를 보여줍니다.

수동 번역 관리의 필요성을 제거하고 맥락에 기반한 즉시 번역을 가능하게 함으로써, Instant18n은 개발자들의 상당한 시간과 노력을 절약해줍니다. 이를 통해 개발자들은 현지화 측면이 원활하고 정확하게 처리되는 동안 애플리케이션의 핵심 기능 구축에 집중할 수 있습니다.

사용자 테스트와 피드백의 중요성

마지막으로, 사용자 테스트와 피드백의 중요성을 항상 명심하세요. 맥락 기반 콘텐츠 생성이 사용자의 기대를 충족하고 애플리케이션의 목표와 부합하는지 검증하는 것이 매우 중요합니다. 사용자의 통찰과 분석을 바탕으로 생성된 콘텐츠를 지속적으로 개선하고 수정하세요. 만약 당신과 팀이 수동으로 검증하기 불가능한 대규모의 동적 콘텐츠를 생성하고 있다면, 사용자가 이상하거나 잘못된 콘텐츠를 그 이유와 함께 신고할 수 있는 피드백 메커니즘을 추가하는 것을 고려해보세요. 그 귀중한 피드백은 심지어 해당 콘텐츠를 생성한 구성 요소를 조정하는 AI 작업자에게 전달될 수도 있습니다!

생성형 UI



오늘날 주목도가 매우 중요해짐에 따라, 효과적인 사용자 참여를 위해서는 단순히 원활하고 직관적인 것을 넘어서 개인의 필요, 선호도, 상황에 맞게 고도로 개인화된 소프트웨어 경험이 필요하게 되었습니다. 그 결과, 디자이너와 개발자들은 각 사용자의 고유한 요구사항에 대규모로 적응하고 부응할 수 있는 사용자 인터페이스를 만들어야 하는 과제에 직면하고 있습니다.

생성형 UI (GenUI)는 사용자 인터페이스 설계에 있어 진정으로 혁신적인 접근 방식입니다. 이는 대규모 언어 모델(LLM)의 능력을 활용하여 실시간으로 고도로 개인화되고 동적인 사용자 경험을 만들어냅니다. 이 책에서 GenUI에 대한 기초적인 내용을 다루고자 하는 이유는, 이것이 애플리케이션 설계와 프레임워크 영역에서 현존하는 가장 유망한 기회 중 하나라고 믿기 때문입니다. 이 특정 분야에서 수십 개 이상의 새로운 상업적, 오픈소스 프로젝트들이 등장할 것이라고 확신합니다.

GenUI의 핵심은 **맥락 기반 콘텐츠 생성**의 원칙을 고급 AI 기술과 결합하여, 사용자의 맥락, 선호도, 목표에 대한 깊은 이해를 바탕으로 텍스트, 이미지, 레이아웃과 같은 사용자 인터페이스 요소를 동적으로 생성하는 것입니다. GenUI를 통해 디자이너와 개발자들은 사용자 상호작용에 반응하여 적응하고 발전하는 인터페이스를 만들 수 있으며, 이는 이전에는 달성할 수 없었던 수준의 개인화를 제공합니다.

GenUI는 사용자 인터페이스 설계 방식의 근본적인 변화를 의미합니다. 대중을 위한 설계 대신, GenUI는 개인을 위한 설계를 가능하게 합니다. 개인화된 콘텐츠와 인터페이스는 각 사용자와 더 깊은 수준에서 공명하는 사용자 경험을 만들어내어, 참여도, 만족도, 충성도를 높일 수 있는 잠재력을 가지고 있습니다.

최첨단 기술로서, GenUI로의 전환에는 많은 개념적, 실용적 도전과제가 있습니다. 설계 과정에 AI를 통합하고, 생성된 인터페이스가 개인화될 뿐만 아니라 사용 가능하고 접근성이 있으며 전반적인 브랜드와 사용자 경험에 부합하도록 보장하는 것, 이 모든 것이 GenUI를 소수만의 추구 대상으로 만드는 과제입니다. 또한, AI의 개입은 데이터 프라이버시, 투명성, 그리고 심지어 윤리적 함의에 대한 질문을 제기합니다.

이러한 과제에도 불구하고, 대규모 개인화 경험은 우리가 디지털 제품 및 서비스와 상호작용하는 방식을 완전히 변화시킬 수 있는 힘을 가지고 있습니다. 이는 사용자의 능력, 배경, 선호도에 관계없이 다양한 사용자의 요구를 충족시키는 포용적이고 접근 가능한 인터페이스를 만들 수 있는 가능성은 열어줍니다.

이 장에서는 GenUI의 개념을 탐구하며, 몇 가지 정의적 특성, 주요 이점, 잠재적 과제들을 살펴볼 것입니다. 우리는 GenUI의 가장 기본적이고 접근하기 쉬운 형태인 전통적으로 설계되고 구현된 사용자 인터페이스를 위한 텍스트 카피 생성부터 시작하겠습니다.

사용자 인터페이스를 위한 카피 생성

양식 레이블, 툴팁, 설명 텍스트와 같이 애플리케이션의 크롬에 존재하는 테스트 요소들은 일반적으로 템플릿이나 UI 컴포넌트에 하드코딩되어 있어, 모든 사용자에게 일관된 but 일반적인 경험을 제공합니다. 맥락 기반 콘텐츠 생성 패턴을 사용하면 이러한 정적 요소들을 동적이고, 맥락을 인식하며, 개인화된 컴포넌트로 변환할 수 있습니다.

개인화된 양식

양식은 웹과 모바일 애플리케이션의 보편적인 부분으로, 사용자 입력을 수집하는 주요 수단입니다. 하지만 전통적인 양식은 종종 사용자의 특정 맥락이나 필요와 항상 일치하지 않는 표준 레이블과 필드로 일반적이고 비개인적인 경험을 제공합니다. 사용자들은 자신의 필요와 선호도에 맞춤화된 양식을 더 잘 작성하는 경향이 있어, 이는 더 높은 전환율과 사용자 만족도로 이어집니다.

하지만 개인화와 일관성 사이의 균형을 맞추는 것이 중요합니다. 양식을 개별 사용자에 맞게 조정하는 것이 유익할 수 있지만, 친숙함과 예측 가능성의 수준을 유지하는 것이 중요합니다. 사용자는 개인화된 요소가 있더라도 여전히 양식을 쉽게 인식하고 탐색할 수 있어야 합니다.

다음은 영감을 줄 수 있는 몇 가지 개인화된 양식 아이디어입니다:

맥락 기반 필드 제안

GenUI는 사용자의 이전 상호작용, 선호도, 데이터를 분석하여 예측으로서 가능한 필드 제안을 제공할 수 있습니다. 예를 들어, 사용자가 이전에 배송 주소를 입력한 경우, 양식은 저장된 정보로 관련 필드를 자동으로 채울 수 있습니다. 이는 시간을 절약할 뿐만 아니라 애플리케이션이 사용자의 선호도를 이해하고 기억한다는 것을 보여줍니다.

잠깐만요, 이런 기술은 AI 없이도 구현할 수 있지 않나요? 물론 가능합니다. 하지만 이러한 기능을 AI로 구동하는 것의 장점은 크게 두 가지입니다: 1) 구현이 매우 쉽다는 점과 2) UI가 변경되고 발전해도 유연하게 대응할 수 있다는 점입니다.

이제 우리의 이론적인 주문 처리 시스템을 위해, 사용자에게 알맞은 배송 주소를 선제적으로 채워주는 서비스를 만들어보겠습니다.

```
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
```

이 예제는 매우 단순화되어 있지만, 대부분의 경우에 작동할 것입니다. 핵심 아이디어는 AI가 사람처럼 추측을 하도록 하는 것입니다. 제가 말씀드리는 내용을 명확하게 하기 위해, 다음과 같은 샘플 데이터를 살펴보겠습니다:

```
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 ]
```

데이터에서 패턴을 발견하셨나요? 이것은 LLM에게 정말 쉬운 일이라고 장담 할 수 있습니다. 이를 증명하기 위해, GPT-4에게 “온도계“의 가장 가능성 있는 배송 주소가 무엇인지 물어보겠습니다.

- 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.

만약 이런 작업에 GPT-4와 같은 비싼 모델을 사용하는 것이 과도하다고 생각
하신다면, 정답입니다! 제가 같은 프롬프트를 Mistral 7B Instruct로 시도해보았
더니 초당 75개의 토큰을 처리하면서 겨우 0.000218 USD의 비용으로 다음과
같은 응답을 생성했습니다.

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.

이러한 기술의 오버헤드와 비용이 체크아웃 경험을 더 마법같이 만들 만한 가치가 있을까요? 많은 온라인 소매업체들에게는 확실히 그렇습니다. 그리고 현재 상황을 보면, AI 컴퓨팅 비용은 계속 하락할 것으로 보입니다. 특히 최저가 경쟁을 벌이고 있는 범용 오픈소스 모델 호스팅 제공업체들의 경우에는 더욱 그렇죠.



프롬프트 템플릿과 구조화된IO, 그리고 응답 펜싱을 함께 사용하여 이러한 종류의 채팅 완성을 최적화하세요.

적응형 필드 순서 배치

폼 필드가 표시되는 순서는 사용자 경험과 완료율에 상당한 영향을 미칠 수 있습니다. GenUI를 사용하면 사용자의 맥락과 각 필드의 중요도에 따라 필드 순서를 동적으로 조정할 수 있습니다. 예를 들어, 사용자가 피트니스 앱의 등록 양식을 작성하는 경우, 양식은 사용자의 피트니스 목표와 선호도와 관련된 필드에 우선순위를 둠으로써 프로세스를 더 관련성 있고 매력적으로 만들 수 있습니다.

개인화된 마이크로카피

GenUI를 사용하여 양식과 관련된 안내 텍스트, 오류 메시지 및 기타 마이크로 카피도 개인화할 수 있습니다. “잘못된 이메일 주소입니다”와 같은 일반적인 오류 메시지 대신, “주문 확인을 받으시려면 유효한 이메일 주소를 입력해 주세요”와 같은 더 도움이 되고 맥락에 맞는 메시지를 생성할 수 있습니다. 이러한

개인화된 요소들은 양식 작성 경험을 더 사용자 친화적이고 덜 답답하게 만들 수 있습니다.

개인화된 유효성 검사

개인화된 마이크로카피와 같은 맥락에서, AI를 사용하여 마법같이 느껴지는 방식으로 양식을 검증할 수 있습니다. AI가 의미론적 수준에서 잠재적 실수를 찾아내며 사용자 프로필 양식을 검증하는 것을 상상해 보세요.

Create your account

Full name

Obie Fernandez

Email

obiefenandez@gmail.com



Did you mean obiefernandez@gmail.com? [Yes, update.](#)

Country

United States



Password

.....



Nice work. This is an excellent password.

그림 8. 의미론적 유효성 검사가 진행되고 있는 것을 발견하셨나요?

점진적 공개

GenUI는 사용자의 맥락에 따라 필수적인 양식 필드를 지능적으로 결정하고 필요에 따라 추가 필드를 점진적으로 표시할 수 있습니다. 이러한 점진적 공개 기법은 인지 부하를 줄이고 양식 작성 과정을 더 관리하기 쉽게 만듭니다. 예를 들어, 사용자가 기본 구독을 신청하는 경우 양식은 처음에 필수 필드만 표시

하고, 사용자가 진행하거나 특정 옵션을 선택함에 따라 관련된 추가 필드를 동적으로 도입할 수 있습니다.

맥락 인식 설명 텍스트

툴팁은 사용자가 특정 요소 위에 마우스를 올리거나 상호작용할 때 추가 정보나 안내를 제공하는 데 자주 사용됩니다. “맥락적 콘텐츠 생성” 접근 방식을 사용하면 사용자의 맥락에 맞춰 적응하고 관련 정보를 제공하는 툴팁을 생성할 수 있습니다. 예를 들어, 사용자가 복잡한 기능을 탐색하는 경우 툴팁은 이전 상호작용이나 숙련도를 기반으로 개인화된 팁이나 예시를 제공할 수 있습니다.

설명 텍스트(지침, 설명 또는 도움말 메시지 등)는 사용자의 맥락에 따라 동적으로 생성될 수 있습니다. 일반적인 설명을 제시하는 대신 LLM을 사용하여 사용자의 특정 요구사항이나 질문에 맞춤화된 텍스트를 생성할 수 있습니다. 예를 들어, 사용자가 특정 단계에서 어려움을 겪고 있다면, 설명 텍스트는 개인화된 안내나 문제 해결 팁을 제공할 수 있습니다.

마이크로카피는 버튼 레이블, 오류 메시지 또는 확인 프롬프트와 같이 사용자를 애플리케이션 전반에 걸쳐 안내하는 작은 텍스트를 말합니다. [맥락적 콘텐츠 생성](#) 접근 방식을 마이크로카피에 적용함으로써, 사용자의 행동에 반응하고 관련성 있고 도움이 되는 텍스트를 제공하는 적응형 UI를 만들 수 있습니다. 예를 들어, 사용자가 중요한 작업을 수행하려고 할 때, 확인 프롬프트는 명확하고 개인화된 메시지를 동적으로 생성할 수 있습니다.

개인화된 설명 텍스트와 툴팁은 새로운 사용자의 온보딩 프로세스를 크게 향상시킬 수 있습니다. 맥락에 맞는 안내와 예시를 제공함으로써, 사용자가 애플리케이션을 빠르게 이해하고 탐색할 수 있도록 도와주어 학습 곡선을 줄이고 채택률을 높일 수 있습니다.

동적이고 맥락을 인식하는 크롬 요소들은 또한 애플리케이션을 더 직관적이고 매력적으로 만들 수 있습니다. 동반되는 텍스트가 사용자의 특정 요구사항과 관심사에 맞춰져 있을 때, 사용자들은 기능들과 더 적극적으로 상호작용하고

탐색할 가능성이 높습니다.

지금까지 AI로 기존 UI 패러다임을 향상시키는 아이디어들을 다뤄왔지만, 더 급진적인 방식으로 사용자 인터페이스의 설계와 구현 방식을 재고해보면 어떨까요?

생성형 UI 정의하기

디자이너가 고정되고 정적인 인터페이스를 만드는 전통적인 UI 설계와 달리, GenUI는 실시간으로 진화하고 적응할 수 있는 유연하고 개인화된 경험을 제공하는 소프트웨어의 미래를 암시합니다. AI 기반 대화형 인터페이스를 사용할 때마다, 우리는 AI가 사용자의 특정 요구사항에 적응하도록 허용하고 있습니다. GenUI는 이러한 수준의 적응성을 소프트웨어의 시각적 인터페이스에 적용함으로써 한 단계 더 나아갑니다.

오늘날 GenUI 아이디어를 실험해볼 수 있는 이유는 대규모 언어 모델들이 이미 프로그래밍을 이해하고 있으며 UI 기술과 프레임워크에 대한 기본 지식을 포함하고 있기 때문입니다. 따라서 핵심 질문은 대규모 언어 모델을 사용하여 각 개별 사용자에게 맞춤화된 텍스트, 이미지, 레이아웃, 심지어 전체 인터페이스와 같은 UI 요소들을 생성할 수 있는지입니다. 모델은 사용자의 과거 상호작용, 명시된 선호도, 인구통계 정보, 현재 사용 맥락 등 다양한 요소들을 고려하여 고도로 개인화되고 관련성 있는 인터페이스를 만들도록 지시받을 수 있습니다.

GenUI는 전통적인 사용자 인터페이스 설계와 다음과 같은 주요 차이점이 있습니다:

1. 동적이고 적응적: 전통적인 UI 설계는 모든 사용자에게 동일하게 유지되는 고정된 정적 인터페이스를 만드는 것을 포함합니다. 반면에 GenUI는 사용자의 요구사항과 맥락에 따라 동적으로 적응하고 변화할 수 있는

인터페이스를 가능하게 합니다. 이는 동일한 애플리케이션이 서로 다른 사용자에게, 또는 같은 사용자라도 다른 상황에서 서로 다른 인터페이스를 제공할 수 있다는 것을 의미합니다.

2. 대규모 개인화: 전통적인 설계에서는 시간과 자원의 제약으로 인해 각 사용자에게 개인화된 경험을 만드는 것이 종종 비현실적입니다. 반면에 GenUI는 대규모 개인화를 가능하게 합니다. AI를 활용함으로써, 디자이너들은 각 사용자 세그먼트에 대해 별도의 인터페이스를 수동으로 설계하고 개발할 필요 없이 각 사용자의 고유한 요구사항과 선호도에 자동으로 적응하는 인터페이스를 만들 수 있습니다.
3. 결과 중심: 전통적인 UI 설계는 종종 시각적으로 매력적이고 기능적인 인터페이스를 만드는 데 중점을 둡니다. 이러한 측면들이 GenUI에서도 여전히 중요하지만, 주된 초점은 원하는 사용자 결과를 달성하는 것으로 옮겨갑니다. GenUI는 순수하게 미적인 고려사항보다는 사용성과 효과성을 우선시하여 각 사용자의 특정 목표와 작업에 최적화된 인터페이스를 만드는 것을 목표로 합니다.
4. 지속적인 학습과 개선: GenUI 시스템은 사용자 상호작용과 피드백을 기반으로 시간이 지남에 따라 지속적으로 학습하고 개선될 수 있습니다. 사용자들이 생성된 인터페이스와 상호작용할 때, AI 모델들은 사용자 행동, 선호도, 결과에 대한 데이터를 수집하여 이 정보를 future 인터페이스 생성을 개선하고 최적화하는 데 사용할 수 있습니다. 이러한 반복적인 학습 프로세스를 통해 GenUI 시스템은 시간이 지남에 따라 사용자 요구사항을 충족시키는 데 더욱 효과적이 될 수 있습니다.

GenUI는 디자인 제안을 제공하거나 특정 디자인 작업을 자동화하는 것과 같은 AI 지원 디자인 도구와는 다르다는 점을 주목하는 것이 중요합니다. 이러한 도구들이 디자인 프로세스를 간소화하는 데 도움이 될 수 있지만, 여전히 최종 결정을 내리고 정적 인터페이스를 만드는 디자이너에게 의존합니다. 반면에 GenUI는 사용자 데이터와 맥락을 기반으로 인터페이스를 실제로 생성하고 적응시키는 데 AI 시스템이 더 적극적인 역할을 하는 것을 포함합니다.

GenUI는 획일적인 솔루션에서 벗어나 고도로 개인화되고 적응적인 경험으로 나아가는 사용자 인터페이스 설계 접근 방식의 중요한 변화를 나타냅니다. AI의 힘을 활용함으로써, GenUI는 각 개별 사용자에게 더 직관적이고 매력적이며 효과적인 인터페이스를 만들어 디지털 제품 및 서비스와 상호작용하는 방식을 혁신할 잠재력을 가지고 있습니다.

예시

GenUI의 개념을 설명하기 위해, “FitAI”라는 가상의 피트니스 애플리케이션을 고려해보겠습니다. 이 앱은 개인의 목표, 체력 수준, 선호도에 기반하여 개인화된 운동 계획과 영양 조언을 제공하는 것을 목표로 합니다.

전통적인 UI 설계 접근방식에서 FitAI는 모든 사용자에게 동일한 고정된 화면과 요소들을 가질 수 있습니다. 하지만 GenUI를 사용하면, 앱의 인터페이스는 각 사용자의 고유한 요구사항과 맥락에 따라 동적으로 적응할 수 있습니다.

이러한 접근방식은 2024년에 구현하기에는 다소 무리가 있을 수 있으며 충분한 ROI를 얻지 못할 수도 있지만, 가능한 것은 사실입니다.

다음은 이것이 어떻게 작동할 수 있는지 보여줍니다:

1. 온보딩:

- 표준 설문지 대신, FitAI는 대화형 AI를 사용하여 사용자의 목표, 현재 체력 수준, 선호도에 대한 정보를 수집합니다.
- 이 초기 상호작용을 기반으로, AI는 사용자의 목표와 가장 관련 있는 기능과 정보를 강조하는 개인화된 대시보드 레이아웃을 생성합니다.
- 현재 AI 기술은 개인화된 대시보드를 구성하는 데 사용할 수 있는 화면 구성 요소들을 선택할 수 있습니다.
- 미래의 AI 기술은 경험 많은 UI 디자이너의 역할을 맡아 실제로 대시보드를 처음부터 만들 수 있을 것입니다.

2. 운동 계획표:

- AI는 사용자의 경험 수준과 사용 가능한 운동 기구에 맞춰 운동 계획 표 인터페이스를 조정합니다.
- 운동 기구가 없는 초보자의 경우, 자세한 설명과 영상이 포함된 간단한 맨몸 운동을 보여줄 수 있습니다.
- 헬스장을 이용할 수 있는 고급 사용자의 경우, 설명이 적은 더 복잡한 루틴을 표시할 수 있습니다.
- 운동 계획표의 내용은 단순히 큰 데이터 집합에서 필터링된 것이 아닙니다. 사용자에 대해 알려진 모든 맥락을 포함하는 쿼리로 지식 베이스를 검색하여 실시간으로 생성될 수 있습니다.

3. 진행 상황 추적:

- 진행 상황 추적 인터페이스는 사용자의 목표와 참여 패턴을 기반으로 발전합니다.
- 체중 감량에 주로 초점을 맞춘 사용자의 경우, 체중 변화 그래프와 칼로리 소모 통계가 눈에 띄게 표시될 수 있습니다.
- 근육을 키우는 사용자의 경우, 근력 향상과 체성분 변화를 강조할 수 있습니다.
- AI는 애플리케이션의 이 부분을 사용자의 실제 진행 상황에 맞춰 조정할 수 있습니다. 진행이 일정 기간 정체되면, 앱은 그 원인을 파악하고 해결하기 위해 사용자가 setback의 이유를 밝히도록 유도하는 모드로 전환될 수 있습니다.

4. 영양 상담:

- 영양 섹션은 사용자의 식이 선호도와 제한사항에 맞춰 조정됩니다.
- 비건 사용자의 경우, 식물성 식사 제안과 단백질 공급원을 보여줄 수 있습니다.
- 글루텐 불내증이 있는 사용자의 경우, 추천 식품에서 글루텐이 포함된 음식을 자동으로 제외합니다.
- 여기서도 콘텐츠는 모든 사용자에게 적용되는 방대한 식사 데이터의 집합에서 가져오는 것이 아니라, 사용자의 특정 상황과 제약 조건에 따라 조정 가능한 정보를 포함하는 지식 베이스에서 합성됩니다.

- 예를 들어, 레시피는 사용자의 피트니스 수준과 신체 통계가 변화함에 따라 지속적으로 변하는 칼로리 요구량에 맞는 재료 명세와 함께 생성됩니다.

5. 동기부여 요소:

- 앱의 동기부여 콘텐츠와 알림은 사용자의 성격 유형과 다양한 동기 부여 전략에 대한 반응을 기반으로 개인화됩니다.
- 일부 사용자는 격려하는 메시지를 받고, 다른 사용자는 더 데이터 중심의 피드백을 받을 수 있습니다.

이 예시에서 GenUI는 FitAI가 각 사용자를 위한 고도로 맞춤화된 경험을 만들 수 있게 하여, 잠재적으로 참여도, 만족도, 그리고 피트니스 목표 달성을 가능성을 높입니다. 인터페이스 요소, 콘텐츠, 심지어 앱의 “성격”까지도 각 개별 사용자의 필요와 선호도에 가장 잘 부합하도록 조정됩니다.

결과 중심 디자인으로의 전환

GenUI는 사용자 인터페이스 디자인의 접근 방식에서 특정 인터페이스 요소 생성에서 더 총체적이고 결과 중심적인 접근으로의 근본적인 전환을 나타냅니다. 이러한 전환은 몇 가지 중요한 의미를 가집니다:

1. 사용자 목표 중심:

- 디자이너들은 특정 인터페이스 구성 요소보다는 사용자 목표와 원하는 결과에 대해 더 깊이 생각해야 할 것입니다.
- 사용자가 목표를 효율적이고 효과적으로 달성하도록 도울 수 있는 인터페이스를 생성할 수 있는 시스템을 만드는 데 중점을 둘 것입니다.
- 미리 정의된 화면 사양이 아닌, 실시간으로 그리고 처음부터 사용자 경험을 생성할 수 있는 AI 기반 디자이너를 위한 새로운 UI 프레임워크가 등장할 것입니다.

2. 디자이너의 역할 변화:

- 디자이너들은 고정된 레이아웃을 만드는 것에서 AI 시스템이 인터페이스를 생성할 때 따라야 할 규칙, 제약 조건, 가이드라인을 정의하는 것으로 전환될 것입니다.
- GenUI 시스템을 효과적으로 안내하기 위해 데이터 분석, AI 프롬프트 엔지니어링, 시스템 사고 등의 분야에서 기술을 개발해야 할 것입니다.

3. 사용자 리서치의 중요성:

- GenUI 맥락에서 사용자 리서치는 더욱 중요해집니다. 디자이너들은 사용자 선호도뿐만 아니라 이러한 선호도와 요구사항이 다양한 맥락에서 어떻게 변화하는지 이해해야 합니다.
- AI의 효과적인 인터페이스 생성 능력을 개선하기 위해 지속적인 사용자 테스트와 피드백 루프가 필수적일 것입니다.

4. 가변성을 위한 디자인:

- 하나의 “완벽한” 인터페이스를 만드는 대신, 디자이너들은 여러 가능한 변형을 고려하고 시스템이 다양한 사용자 요구에 적합한 인터페이스를 생성할 수 있도록 해야 합니다.
- 여기에는 옛지 케이스를 위한 디자인과 생성된 인터페이스가 다양한 구성에서 사용성과 접근성을 유지하도록 보장하는 것이 포함됩니다.
- 제품 차별화는 사용자 심리학에 대한 다양한 관점과 경쟁사가 접근할 수 없는 고유한 데이터 세트 및 지식 베이스의 활용과 관련된 새로운 차원을 갖게 됩니다.

과제와 고려사항

GenUI는 흥미로운 가능성을 제공하지만, 동시에 몇 가지 과제와 고려사항도 제시합니다:

1. 기술적 한계:

- 현재의 AI 기술은 진보적이지만, 복잡한 사용자 의도를 이해하고 진정한 맥락 인식 인터페이스를 생성하는 데 여전히 한계가 있습니다.
- 특히 성능이 낮은 기기에서 인터페이스 요소를 실시간으로 생성하는 것과 관련된 성능 문제가 있습니다.

2. 데이터 요구사항:

- 사용 사례에 따라 효과적인 GenUI 시스템은 개인화된 인터페이스를 생성하기 위해 상당한 양의 사용자 데이터가 필요할 수 있습니다.
- 윤리적으로 진정성 있는 사용자 데이터를 확보하는 데 따른 과제들은 데이터 프라이버시와 보안, 그리고 GenUI 모델 학습에 사용되는 데이터의 잠재적 편향성에 대한 우려를 제기합니다.

3. 사용성과 일관성:

- 적어도 이 방식이 널리 보급될 때까지는, 지속적으로 변화하는 인터페이스를 가진 애플리케이션은 사용자들이 익숙한 요소를 찾거나 효율적으로 탐색하는데 어려움을 겪을 수 있어 사용성 문제로 이어질 수 있습니다.
- 개인화와 일관되고 학습 가능한 인터페이스 유지 사이의 균형을 맞추는 것이 매우 중요할 것입니다.

4. AI에 대한 과도한 의존:

- 디자인 결정을 AI 시스템에 과도하게 위임하면 영감 없는, 문제가 있는, 또는 단순히 작동하지 않는 인터페이스 선택으로 이어질 수 있는 위험이 있습니다.
- 가까운 미래에도 인간의 감독과 AI가 생성한 디자인을 수정할 수 있는 능력이 여전히 중요할 것입니다.

5. 접근성 문제:

- 동적으로 생성되는 인터페이스가 장애가 있는 사용자들에게도 접근 가능하도록 보장하는 것은 완전히 새로운 과제를 제시하며, 이는 일반적인 시스템에서 보여주는 낮은 수준의 접근성 준수를 고려할 때 우려되는 부분입니다.
- 반면에, AI 디자이너들은 접근성에 대한 내장된 고려사항과 비장애인 사용자를 위한 UI를 구축하는 것처럼 즉석에서 접근 가능한 인터페이스를 구축하는 기능을 갖출 수 있습니다.
- 어떤 경우든, GenUI 시스템은 강력한 접근성 지침과 테스트 프로세스를 갖추고 설계되어야 합니다.

6. 사용자 신뢰와 투명성:

- 사용자들은 자신에 대해 “너무 많이 알고 있는” 것처럼 보이거나 이해할 수 없는 방식으로 변화하는 인터페이스에 불편함을 느낄 수 있습니다.
- 인터페이스가 어떻게, 그리고 왜 개인화되는지에 대한 투명성을 제공하는 것이 사용자 신뢰를 구축하는 데 중요할 것입니다.

미래 전망과 기회

생성형 UI(GenUI)의 미래는 우리가 디지털 제품 및 서비스와 상호작용하는 방식을 혁신할 엄청난 가능성을 품고 있습니다. 이 기술이 계속 발전함에 따라, 사용자 인터페이스가 설계되고, 구현되고, 경험되는 방식에 있어 큰 변화가 있을 것으로 예상됩니다. 저는 GenUI가 우리의 소프트웨어를 현재 공상과학으로 여겨지는 영역으로 마침내 밀어붙일 현상이라고 생각합니다.

GenUI의 가장 흥미로운 전망 중 하나는 단순히 심각한 장애가 있는 사람들이 소프트웨어 사용에서 완전히 배제되지 않도록 하는 것을 넘어서는 대규모의 접근성 향상 잠재력입니다. 인터페이스를 개별 사용자의 요구에 자동으로 적응 시킴으로써, GenUI는 이전보다 더 포용적인 디지털 경험을 만들 수 있습니다.

수동 구성이나 별도의 “접근성” 버전의 애플리케이션 없이도, 어린 사용자나 시각 장애가 있는 사용자를 위해 더 큰 텍스트를 제공하거나 인지 장애가 있는 사용자를 위해 단순화된 레이아웃을 제공하는 인터페이스를 상상해보십시오.

GenUI의 개인화 기능은 다양한 디지털 제품에서 사용자 참여, 만족도, 충성도를 높일 가능성이 있습니다. 인터페이스가 개인의 선호도와 행동에 더 잘 맞춰짐에 따라, 사용자들은 디지털 경험을 더 직관적이고 즐겁게 느낄 것이며, 이는 잠재적으로 기술과의 더 깊고 의미 있는 상호작용으로 이어질 수 있습니다.

GenUI는 또한 새로운 사용자를 위한 온보딩 프로세스를 변화시킬 잠재력이 있습니다. 각 사용자의 전문성 수준에 빠르게 적응하는 직관적이고 개인화된 첫 사용자 경험을 만들도록써, GenUI는 새로운 애플리케이션과 관련된 학습 곡선을 크게 줄일 수 있습니다. 이는 더 빠른 채택률과 새로운 기능을 탐색하는데 있어 증가된 사용자 자신감으로 이어질 수 있습니다.

또 다른 흥미로운 가능성은 GenUI가 각각의 특정 사용 맥락에 최적화하면서도 서로 다른 기기와 플랫폼 전반에 걸쳐 일관된 사용자 경험을 유지할 수 있다는 점입니다. 이는 스마트폰과 태블릿, 데스크톱 컴퓨터, 그리고 증강현실 안경과 같은 새로운 기술에 이르기까지 점점 더 파편화되는 기기 환경에서 일관된 경험을 제공하는 오랜 과제를 해결할 수 있습니다.

GenUI의 데이터 기반 특성은 UI 디자인의 빠른 반복과 개선을 위한 기회를 제공합니다. 사용자들이 생성된 인터페이스와 상호작용하는 방식에 대한 실시간 데이터를 수집함으로써, 디자이너와 개발자들은 사용자 행동과 선호도에 대한 전례 없는 통찰을 얻을 수 있습니다. 이러한 피드백 루프는 가정이나 제한된 사용자 테스트가 아닌 실제 사용 패턴에 기반한 UI 디자인의 지속적인 개선으로 이어질 수 있습니다.

이러한 변화에 대비하기 위해, 디자이너들은 자신의 기술과 사고방식을 발전 시켜야 할 것입니다. 고정된 레이아웃을 만드는 것에서 AI 기반 인터페이스 생성을 위한 포괄적인 디자인 시스템과 지침을 개발하는 것으로 초점이 이동할 것입니다. 디자이너들은 GenUI 시스템을 효과적으로 안내하기 위해 데이터 분석, AI 기술, 시스템 사고에 대한 깊은 이해를 키워야 할 것입니다.

더욱이, GenUI가 디자인과 기술 사이의 경계를 흐리게 함에 따라, 디자이너들은 개발자 및 데이터 과학자들과 더 긴밀하게 협력해야 할 것입니다. 이러한 학제 간 접근은 시각적으로 매력적이고 사용자 친화적일 뿐만 아니라 기술적으로 견고하고 윤리적으로 건전한 GenUI 시스템을 만드는 데 매우 중요할 것입니다.

기술이 성숙해감에 따라 GenUI의 윤리적 함의도 주목받게 될 것입니다. 디자이너들은 인터페이스 설계에서 AI를 책임감 있게 사용하기 위한 프레임워크를 개발하는 데 중요한 역할을 하게 될 것이며, 이를 통해 개인정보를 침해하거나 사용자의 행동을 비윤리적으로 조작하지 않으면서도 사용자 경험을 향상시킬 수 있도록 보장할 것입니다.

미래를 바라볼 때, GenUI는 흥미진진한 기회와 중요한 도전 과제를 동시에 제시합니다. 이는 전 세계 사용자들에게 더욱 직관적이고 효율적이며 만족스러운 디지털 경험을 제공할 수 있는 잠재력을 가지고 있습니다. 디자이너들이 적응하고 새로운 기술을 습득해야 하겠지만, 동시에 인간-컴퓨터 상호작용의 미래를 깊이 있고 의미 있는 방식으로 형성할 수 있는 전례 없는 기회를 제공합니다. 완전히 실현된 GenUI 시스템을 향한 여정은 분명 복잡할 것이지만, 향상된 사용자 경험과 디지털 접근성 측면에서의 잠재적 이점은 우리가 추구할 가치가 있는 미래를 만들어 낼 것입니다.

지능형 워크플로우 오케스트레이션



이 장에서는 지능형 워크플로우 오케스트레이션 접근 방식을 뒷받침하는 주요 원칙과 패턴들을 살펴볼 것입니다. AI를 사용하여 작업을 지능적으로 라우팅하고, 의사결정을 자동화하며, 사용자 행동, 시스템 성능, 비즈니스 규칙과 같은 다양한 요소들을 기반으로 워크플로우를 동적으로 조정하는 방법을 고려할 것입니다. 실용적인 예시와 실제 시나리오를 통해, 애플리케이션 워크플로우를 간소화하고 최적화하는 데 있어 AI의 혁신적인 잠재력을 보여드릴 것입니다.

복잡한 비즈니스 프로세스를 가진 엔터프라이즈 애플리케이션을 구축하든, 동적인 사용자 여정을 가진 소비자 대상 애플리케이션을 구축하든, 이 장에서 논의되는 패턴과 기술들은 전반적인 사용자 경험을 향상시키고 비즈니스 가치를 창출하는 지능적이고 효율적인 워크플로우를 만드는 데 필요한 지식과

도구를 제공할 것입니다.

비즈니스 필요성

워크플로우 관리에 대한 전통적인 접근 방식은 주로 미리 정의된 규칙과 정적 의사결정 트리에 의존하는데, 이는 현대 애플리케이션의 동적인 특성에 대응하기에는 경직되고 유연성이 부족할 수 있습니다.

더욱이, 애플리케이션이 확장되고 동시 사용자 수가 증가함에 따라, 워크플로우는 실시간 데이터와 시스템 성능을 기반으로 자체적으로 적응하고 최적화할 필요가 있을 수 있습니다. 예를 들어, 트래픽이 많은 시간대에는 애플리케이션이 특정 작업의 우선순위를 조정하고, 리소스를 효율적으로 할당하며, 원활한 사용자 경험을 보장하기 위해 워크플로우를 동적으로 조정해야 할 수 있습니다.

이것이 바로 “지능형 워크플로우 오케스트레이션” 접근 방식이 필요한 이유입니다. AI 컴포넌트를 활용함으로써, 개발자들은 지능적이고 적응적이며 자체 최적화되는 워크플로우를 만들 수 있습니다. AI는 방대한 양의 데이터를 분석하고, 과거 경험으로부터 학습하며, 워크플로우를 효과적으로 조율하기 위해 실시간으로 정보에 기반한 결정을 내릴 수 있습니다.

주요 이점

- 향상된 효율성:** AI는 작업 할당, 리소스 활용, 워크플로우 실행을 최적화하여 더 빠른 처리 시간과 전반적인 효율성 향상을 이끌어낼 수 있습니다.
- 적응성:** AI 기반 워크플로우는 사용자 수요, 시스템 성능, 비즈니스 요구 사항의 변화와 같은 변화하는 조건에 동적으로 적응하여 애플리케이션이 반응적이고 탄력적으로 유지되도록 보장합니다.
- 자동화된 의사결정:** AI는 워크플로우 내의 복잡한 의사결정 프로세스를 자동화하여 수동 개입을 줄이고 인적 오류의 위험을 최소화할 수 있습니다.

4. 개인화: AI는 사용자 행동, 선호도, 컨텍스트를 분석하여 워크플로우를 개인화하고 개별 사용자에게 맞춤형 경험을 제공할 수 있습니다.
5. 확장성: AI 기반 워크플로우는 성능이나 신뢰성을 저해하지 않으면서 증가하는 데이터와 사용자 상호작용의 볼륨을 원활하게 처리할 수 있도록 확장될 수 있습니다.

다음 섹션에서는 지능형 워크플로우 구현을 가능하게 하는 주요 패턴과 기술들을 살펴보고, 현대적 애플리케이션에서 AI가 워크플로우 관리를 어떻게 변화시키고 있는지 실제 사례를 통해 보여드릴 것입니다.

주요 패턴

애플리케이션에서 지능형 워크플로우 오케스트레이션을 구현하기 위해, 개발자들은 AI의 힘을 활용하는 몇 가지 주요 패턴을 활용할 수 있습니다. 이러한 패턴들은 워크플로우를 설계하고 관리하는 데 구조화된 접근 방식을 제공하여, 실시간 데이터와 컨텍스트를 기반으로 프로세스를 적응, 최적화, 자동화할 수 있게 합니다. 지능형 워크플로우 오케스트레이션의 기본적인 패턴들을 살펴보겠습니다.

동적 작업 라우팅

이 패턴은 작업 우선순위, 리소스 가용성, 시스템 성능과 같은 다양한 요소들을 기반으로 워크플로우 내에서 작업을 지능적으로 라우팅하기 위해 AI를 사용하는 것을 포함합니다. AI 알고리즘은 각 작업의 특성을 분석하고, 시스템의 현재 상태를 고려하며, 가장 적절한 리소스나 처리 경로에 작업을 할당하기 위한 정보에 기반한 결정을 내릴 수 있습니다. 동적 작업 라우팅은 작업이 효율적으로 분배되고 실행되도록 보장하여 전반적인 워크플로우 성능을 최적화합니다.

```
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
```

29번 줄의 `while` 표현식으로 생성된 루프를 주목해보세요. 이 루프는 AI에 작업이 할당될 때까지 계속해서 프롬프트를 실행합니다. 35번 줄에서는 나중에 분석과 디버깅이 필요할 경우를 대비해 작업의 기록을 저장합니다.

맥락 기반 의사결정

워크플로우 내에서 맥락을 인식하는 의사결정을 하기 위해 매우 유사한 코드를 사용할 수 있습니다. 사용자 선호도, 과거 패턴, 실시간 입력과 같은 관련 데이터 포인트를 분석함으로써, AI 컴포넌트는 워크플로우의 각 의사결정 시점에서 가장 적절한 행동 방침을 결정할 수 있습니다. 각 사용자나 시나리오의 특정 맥락에 따라 워크플로우의 동작을 조정하여 개인화되고 최적화된 경험을 제공합니다.

적응형 워크플로우 구성

이 패턴은 변화하는 요구사항이나 조건에 기반하여 워크플로우를 동적으로 구성하고 조정하는 데 중점을 둡니다. AI는 워크플로우의 현재 상태를 분석하고, 병목 현상이나 비효율성을 식별하며, 성능을 최적화하기 위해 워크플로우 구조를 자동으로 수정할 수 있습니다. 적응형 워크플로우 구성을 통해 애플리케이션은 수동 개입 없이도 지속적으로 진화하고 프로세스를 개선할 수 있습니다.

예외 처리와 복구

예외 처리와 복구는 지능형 워크플로우 오케스트레이션의 중요한 측면입니다. AI 컴포넌트와 복잡한 워크플로우를 다룰 때는 시스템의 안정성과 신뢰성을 보장하기 위해 예외를 예측하고 우아하게 처리하는 것이 필수적입니다.

다음은 지능형 워크플로우에서 예외 처리와 복구를 위한 주요 고려사항과 기술입니다:

- 예외 전파:** 워크플로우 컴포넌트 전반에 걸쳐 예외를 전파하는 일관된 접근 방식을 구현하세요. 컴포넌트 내에서 예외가 발생하면, 이를 포착하고, 기록하며, 오케스트레이터나 예외 처리를 담당하는 독립 컴포넌트로 전파해야 합니다. 이는 예외 처리를 중앙화하고 예외가 묵살되는 것을 방지하며, [지능형 오류 처리](#)의 가능성을 열어주는 것이 목적입니다.
- 재시도 메커니즘:** 재시도 메커니즘은 워크플로우의 복원력을 향상시키고 일시적인 실패를 우아하게 처리하는 데 도움이 됩니다. 네트워크 연결 문제나 지정된 지연 후 자동으로 재시도할 수 있는 리소스 사용 불가와 같은 일시적이거나 복구 가능한 예외에 대해 반드시 재시도 메커니즘을 구현하세요. AI 기반 오케스트레이터나 예외 처리기를 사용하면 지수 백오프와 같은 고정된 알고리즘에 의존하는 기계적인 재시도 전략을 사용할 필요가 없습니다. 예외 처리 방법을 결정하는 것을 AI 컴포넌트의 “판단”에 맡길 수 있습니다.
- 대체 전략:** AI 컴포넌트가 유효한 응답을 제공하지 못하거나 오류가 발생할 경우(최신 기술의 특성상 흔한 일입니다), 워크플로우가 계속 진행될 수 있도록 대체 메커니즘을 마련하세요. 이는 기본값 사용, 대체 알고리즘, 또는 의사결정을 하고 워크플로우를 진행시키기 위한 [인간 참여 루프](#)를 포함할 수 있습니다.
- 보상 조치:** 오케스트레이터의 지시에는 자동으로 해결할 수 없는 예외를 처리하기 위한 보상 조치에 대한 지침이 포함되어야 합니다. 보상 조치는 실패한 작업의 효과를 취소하거나 완화하기 위해 취하는 단계입니다. 예를

들어, 결제 처리 단계가 실패하면 보상 조치로 거래를 롤백하고 사용자에게 알릴 수 있습니다. 보상 조치는 예외가 발생했을 때 데이터의 일관성과 무결성을 유지하는 데 도움이 됩니다.

5. 예외 모니터링 및 경고: 중요한 예외를 감지하고 관련 이해관계자에게 알리기 위한 모니터링 및 경고 메커니즘을 설정하세요. 오케스트레이터는 예외가 특정 한계를 초과하거나 특정 유형의 예외가 발생할 때 경고를 트리거하는 임계값과 규칙을 인식하도록 설정할 수 있습니다. 이를 통해 문제가 전체 시스템에 영향을 미치기 전에 선제적으로 식별하고 해결할 수 있습니다.

다음은 Ruby 워크플로우 컴포넌트에서의 예외 처리와 복구 예시입니다:

```
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
```

이 예시에서 `InventoryManager` 컴포넌트는 주어진 주문에 대한 제품의 가용성을 확인합니다. 사용 가능한 수량이 부족할 경우, `InsufficientInventoryError`를 발생시킵니다. 이 예외는 포착되어 기록되며, 재시도 메커니즘이 구현됩니다. 재시도 제한을 초과하면, 관리자에게 알림으로써 수동 개입으로 대체됩니다.

강력한 예외 처리 및 복구 메커니즘을 구현함으로써, 지능형 워크플로우가 복원력이 있고, 유지보수가 가능하며, 예상치 못한 상황을 우아하게 처리할 수 있도록 보장할 수 있습니다.

이러한 패턴들은 지능형 워크플로우 오케스트레이션의 기초를 형성하며, 다양한 애플리케이션의 특정 요구사항에 맞게 결합하고 조정할 수 있습니다. 개발자들은 이러한 패턴들을 활용하여 유연하고 복원력이 있으며 성능과 사용자 경험에 최적화된 워크플로우를 만들 수 있습니다.

다음 섹션에서는 이러한 패턴들을 실제로 어떻게 구현할 수 있는지 살펴보고, 실제 예시와 코드 스ニ펫을 통해 AI 컴포넌트를 워크플로우 관리에 통합하는 방법을 설명하겠습니다.

실전에서의 지능형 워크플로우 오케스트레이션 구현

이제 지능형 워크플로우 오케스트레이션의 주요 패턴들을 살펴보았으니, 이러한 패턴들을 실제 애플리케이션에서 어떻게 구현할 수 있는지 자세히 알아보겠습니다. AI 컴포넌트를 워크플로우 관리에 통합하는 것을 설명하기 위해 실제 예시와 코드 스ニ펫을 제공하겠습니다.

지능형 주문 처리자

Ruby on Rails 전자상거래 애플리케이션에서 AI 기반 `OrderProcessor` 컴포넌트를 사용하여 지능형 워크플로우 오케스트레이션을 구현하는 실제 예시를 살펴보겠습니다. `OrderProcessor`는 3장에서 [다수의 작업자](#)를 논의할 때 처음 접했던 [프로세스 관리자 엔터프라이즈 통합](#) 개념을 구현합니다. 이 컴포넌트는 주문 처리 워크플로우를 관리하고, 중간 결과를 기반으로 라우팅 결정을 내리며, 다양한 처리 단계의 실행을 조율하는 역할을 담당합니다.

주문 처리 과정에는 주문 유효성 검사, 재고 확인, 결제 처리, 배송과 같은 여러 단계가 포함됩니다. 각 단계는 특정 작업을 수행하고 결과를 `OrderProcessor`에 반환하는 별도의 작업자 프로세스로 구현됩니다. 이러한 단계들은 필수적이지 않으며, 반드시 정확한 순서로 수행될 필요도 없습니다.

다음은 `OrderProcessor`의 구현 예시입니다. [Raix](#)에서 가져온 두 개의 믹스인을 특징으로 합니다. 첫 번째(`ChatCompletion`)는 채팅 완성 기능을 제공하여 이를 AI 컴포넌트로 만들어줍니다. 두 번째(`FunctionDispatch`)는 AI가 함수를 호출할 수 있게 해주어, 프롬프트에 대해 텍스트 메시지 대신 함수 호출로 응답 할 수 있게 합니다.

작업자 함수들(`validate_order`, `check_inventory` 등)은 각각의 작업자 클래스에 위임하며, 이는 AI 또는 비AI 컴포넌트일 수 있고, 유일한 요구사항은 작업 결과를 문자열로 표현할 수 있는 형식으로 반환하는 것입니다.



이 책의 이 부분의 다른 모든 예시와 마찬가지로, 이 코드는 실질적으로 의사 코드이며 패턴의 의미를 전달하고 여러분만의 창작을 영감을 주기 위한 것입니다. 패턴에 대한 전체 설명과 완전한 코드 예시는 제2부에 포함되어 있습니다.

```
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
```

이 예시에서 OrderProcessor는 주문 객체로 초기화되며, 대규모 언어 모델의 고유한 대화 기록 형식으로 워크플로우 실행 기록을 유지합니다. AI는 주문 유효성 검사, 재고 확인, 결제 처리, 배송과 같은 다양한 처리 단계의 실행을 조율하는 데 완전한 제어권을 가집니다.

chat_completion 메서드가 호출될 때마다 기록이 AI에 전송되어 함수 호출 형태로 완성을 제공합니다. 이전 단계의 결과를 분석하고 적절한 조치를 결정하는 것은 전적으로 AI의 몫입니다. 예를 들어, 재고 확인 결과 재고 수준이 낮은 것으로 나타나면 OrderProcessor는 보충 작업을 예약할 수 있습니다. 결제 처리가 실패하면 재시도를 시작하거나 고객 지원팀에 알릴 수 있습니다.

위의 예시에는 보충이나 고객 지원팀 알림을 위한 함수가 정의되어 있지 않지만, 충분히 구현할 수 있습니다.

함수가 호출될 때마다 기록이 늘어나며, 각 단계의 결과와 다음 단계를 위한 AI 생성 지침을 포함하여 워크플로우 실행의 기록 역할을 합니다. 이 기록은 디버깅, 감사 및 주문 이행 프로세스의 가시성 제공에 활용될 수 있습니다.

OrderProcessor에서 AI를 활용함으로써 전자상거래 애플리케이션은 실시간 데이터를 기반으로 워크플로우를 동적으로 조정하고 예외 상황을 지능적으로 처리할 수 있습니다. AI 구성 요소는 정보에 기반한 의사결정을 내리고, 워크

플로우를 최적화하며, 복잡한 시나리오에서도 원활한 주문 처리를 보장할 수 있습니다.

작업자 프로세스에 대한 유일한 요구사항이 AI가 다음에 무엇을 할지 결정할 때 고려할 수 있는 이해 가능한 출력을 반환하는 것이라는 사실을 통해, 이러한 접근 방식이 서로 다른 시스템을 통합할 때 일반적으로 필요한 입력/출력 매핑 작업을 어떻게 줄일 수 있는지 깨닫기 시작할 것입니다.

지능형 콘텐츠 관리자

소셜 미디어 애플리케이션은 일반적으로 안전하고 건전한 커뮤니티를 보장하기 위해 최소한의 콘텐츠 관리가 필요합니다. 이 예시의 **ContentModerator** 구성 요소는 AI를 활용하여 콘텐츠의 특성과 다양한 관리 단계의 결과를 기반으로 결정을 내리며 관리 워크플로우를 지능적으로 조율합니다.

관리 프로세스는 텍스트 분석, 이미지 인식, 사용자 평판 평가, 수동 검토와 같은 여러 단계를 포함합니다. 각 단계는 특정 작업을 수행하고 결과를 **ContentModerator**에 반환하는 별도의 작업자 프로세스로 구현됩니다.

다음은 **ContentModerator**의 구현 예시입니다:

```
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
```

이 예시에서 ContentModerator는 콘텐츠 객체로 초기화되며 대화 형식으로 모더레이션 기록을 유지합니다. AI 구성 요소는 콘텐츠의 특성과 각 단계의 결과를 기반으로 실행할 단계를 결정하는 모더레이션 워크플로우를 완전히 제어합니다.

AI가 호출할 수 있는 워커 함수에는 `analyze_text`, `recognize_image`, `assess_user_reputation`, 그리고 `escalate_to_manual_review`가 포함됩니다. 각 함수는 해당 작업을 관련 워커 프로세스(`TextAnalysisWorker`, `ImageRecognitionWorker` 등)에 위임하고 그 결과를 모더레이션 기록에 추가합니다. 단, 에스컬레이션 함수는 최종 상태로 작동합니다. 마지막으로, `approve_content`와 `reject_content` 함수도 최종 상태로 작동합니다.

AI 구성 요소는 콘텐츠를 분석하고 적절한 조치를 결정합니다. 콘텐츠에 이미지 참조가 포함되어 있다면, 시각적 검토를 위해 `recognize_image` 워커를 호출할 수 있습니다. 어떤 워커가 잠재적으로 유해한 콘텐츠를 경고하면, AI는 해당 콘텐츠를 수동 검토로 에스컬레이션하거나 즉시 거부하기로 결정할 수 있습니다. 하지만 경고의 심각성에 따라, AI는 확신하기 어려운 콘텐츠를 처리하는 방법을 결정할 때 사용자 평판 평가 결과를 활용할 수 있습니다. 사용 사례에 따라, 신뢰할 수 있는 사용자에게는 게시할 수 있는 콘텐츠의 범위가 더 넓을 수 있습니다. 이런 식으로 계속됩니다…

이전 프로세스 관리자 예시와 마찬가지로, 모더레이션 기록은 각 단계의 결과와 AI가 생성한 결정을 포함하여 워크플로우 실행의 기록 역할을 합니다. 이 기록은 감사, 투명성, 그리고 시간이 지남에 따라 모더레이션 프로세스를 개선하는데 사용될 수 있습니다.

`ContentModerator`에서 AI를 활용함으로써, 소셜 미디어 애플리케이션은 콘텐츠의 특성에 따라 모더레이션 워크플로우를 동적으로 조정하고 복잡한 모더레이션 시나리오를 지능적으로 처리할 수 있습니다. AI 구성 요소는 정보에 기반한 결정을 내리고, 워크플로우를 최적화하며, 안전하고 건강한 커뮤니티 경험을 보장할 수 있습니다.

지능형 워크플로우 오케스트레이션의 맥락에서 예측적 작업 스케줄링과 예외 처리 및 복구를 보여주는 두 가지 예시를 더 살펴보겠습니다.

고객 지원 시스템의 예측적 작업 스케줄링

```
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
```

이 예시에서 `SupportTicketScheduler`는 지원 티켓 객체로 초기화되며 스케줄링 기록을 유지합니다. AI 구성 요소는 티켓 긴급도, 상담원 전문성, 예측된 상담원 업무량과 같은 요소들을 기반으로 티켓 배정을 예측적으로 스케줄링하기

위해 티켓 세부 사항을 분석합니다.

AI가 호출할 수 있는 함수들에는 `analyze_ticket_urgency`, `list_available_agents`, `predict_agent_workload`, `assign_ticket_to_agent`가 포함됩니다. 각 함수는 해당 분석기 또는 예측기 구성 요소에 작업을 위임하고 결과를 스케줄링 기록에 추가합니다. AI는 또한 `delay_assignment` 함수를 사용하여 배정을 지연시킬 수 있는 옵션을 가지고 있습니다.

AI 구성 요소는 스케줄링 기록을 검토하고 티켓 배정에 대한 정보에 기반한 결정을 내립니다. 티켓의 긴급도, 사용 상담원의 전문성, 각 상담원의 예측된 업무량을 고려하여 티켓 처리에 가장 적합한 상담원을 결정합니다.

예측적 작업 스케줄링을 활용함으로써, 고객 지원 애플리케이션은 티켓 배정을 최적화하고, 응답 시간을 단축하며, 전반적인 고객 만족도를 향상시킬 수 있습니다. 지원 티켓의 선제적이고 효율적인 관리를 통해 적절한 티켓이 적절한 시기에 적절한 상담원에게 배정되도록 보장합니다.

데이터 처리 파이프라인에서의 예외 처리와 복구

예외를 처리하고 실패로부터 복구하는 것은 데이터 무결성을 보장하고 데이터 손실을 방지하는 데 필수적입니다. `DataProcessingOrchestrator` 구성 요소는 데이터 처리 파이프라인에서 예외를 지능적으로 처리하고 복구 프로세스를 조율하기 위해 AI를 활용합니다.

```
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
```

이 예시에서 DataProcessingOrchestrator는 데이터 배치 객체로 초기화되며 처리 기록을 유지합니다. AI 컴포넌트는 데이터 처리 파이프라인을 조율하며, 필요에 따라 예외를 처리하고 실패로부터 복구합니다.

AI가 호출할 수 있는 함수에는 validate_data, process_data, request_fix, retry_processing, 그리고 mark_data_as_failed가 포함됩니다. 각 함수는 해당 작업을 관련 데이터 처리 컴포넌트에 위임하고 결과 또는 예외 세부 사항을 처리 기록에 추가합니다.

validate_data 단계에서 유효성 검사 예외가 발생하면, handle_validation_exception 함수가 예외 데이터를 기록에 추가하고 제어권을 AI에게 반환합니다. 마찬가지로, process_data 단계에서 처리 예외가 발생하면, AI는 복구 전략을 결정할 수 있습니다.

발생한 예외의 성격에 따라, AI는 자체 판단하에 request_fix를 호출할 수 있

으며, 이는 AI 기반 SmartDataFixer 컴포넌트에 작업을 위임합니다(자가 치유 데이터 장 참조). 데이터 수정기는 @data_batch를 어떻게 수정해야 처리를 재시도할 수 있는지에 대한 일반적인 설명을 받습니다. 아마도 성공적인 재시도를 위해서는 유효성 검사에 실패한 레코드를 데이터 배치에서 제거하거나 사람의 검토를 위해 다른 처리 파이프라인으로 복사하는 것이 필요할 수 있겠죠? 가능성은 거의 무한합니다.

AI 기반 예외 처리와 복구를 통합함으로써, 데이터 처리 애플리케이션은 더욱 탄력적이고 장애에 강해집니다. DataProcessingOrchestrator는 지능적으로 예외를 관리하고, 데이터 손실을 최소화하며, 데이터 처리 워크플로우의 원활한 실행을 보장합니다.

모니터링과 로깅

모니터링과 로깅은 AI 기반 워크플로우 컴포넌트의 진행 상황, 성능, 상태를 파악할 수 있게 해주어 개발자가 시스템의 동작을 추적하고 분석할 수 있게 합니다. 효과적인 모니터링과 로깅 메커니즘을 구현하는 것은 지능형 워크플로우의 디버깅, 감사, 지속적인 개선을 위해 필수적입니다.

워크플로우 진행 상황과 성능 모니터링

지능형 워크플로우의 원활한 실행을 보장하기 위해서는 각 워크플로우 컴포넌트의 진행 상황과 성능을 모니터링하는 것이 중요합니다. 이는 워크플로우 수명 주기 전반에 걸쳐 주요 지표와 이벤트를 추적하는 것을 포함합니다.

모니터링해야 할 중요한 측면들은 다음과 같습니다:

1. 워크플로우 실행 시간: 각 워크플로우 컴포넌트가 작업을 완료하는 데 걸리는 시간을 측정합니다. 이는 성능 병목 현상을 식별하고 전반적인 워크플로우 효율성을 최적화하는 데 도움이 됩니다.

2. 리소스 사용률: CPU, 메모리, 저장소와 같은 시스템 리소스의 각 워크플로우 컴포넌트별 사용률을 모니터링합니다. 이는 시스템이 용량 내에서 운영되고 작업량을 효과적으로 처리할 수 있도록 보장하는 데 도움이 됩니다.
3. 오류율과 예외: 워크플로우 컴포넌트 내의 오류와 예외 발생을 추적합니다. 이는 잠재적인 문제를 식별하고 선제적인 오류 처리와 복구를 가능하게 합니다.
4. 의사결정 지점과 결과: 워크플로우 내의 의사결정 지점과 AI 기반 결정의 결과를 모니터링합니다. 이는 AI 컴포넌트의 동작과 효과성에 대한 통찰을 제공합니다.

모니터링 프로세스로 수집된 데이터는 대시보드에 표시되거나 시스템 관리자에게 시스템 상태를 알리는 예약된 보고서의 입력으로 사용될 수 있습니다.



모니터링 데이터는 검토와 잠재적 조치를 위해 AI 기반 시스템 관리자 프로세스에 제공될 수 있습니다!

주요 이벤트와 결정 로깅

로깅은 워크플로우 실행 중에 발생하는 주요 이벤트, 결정, 예외에 대한 관련 정보를 캡처하고 저장하는 필수적인 관행입니다.

로깅해야 할 중요한 측면들은 다음과 같습니다:

1. 워크플로우 시작과 완료: 각 워크플로우 인스턴스의 시작 및 종료 시간을 입력 데이터와 사용자 컨텍스트와 같은 관련 메타데이터와 함께 기록합니다.
2. 컴포넌트 실행: 입력 매개변수, 출력 결과, 생성된 중간 데이터를 포함하여 각 워크플로우 컴포넌트의 실행 세부 사항을 기록합니다.
3. AI 결정과 추론: 기본 추론이나 신뢰도 점수와 함께 AI 컴포넌트가 내린 결정을 기록합니다. 이는 투명성을 제공하고 AI 기반 결정의 감사를 가능하게 합니다.

4. 예외와 오류 메시지: 스택 트레이스와 관련 컨텍스트 정보를 포함하여 워크 플로우 실행 중 발생한 예외나 오류 메시지를 기록합니다.

로깅은 로그 파일 작성, 데이터베이스에 로그 저장, 중앙 집중식 로깅 서비스로 로그 전송 등 다양한 기술을 사용하여 구현할 수 있습니다. 애플리케이션 아키텍처와 쉽게 통합되고 유연성과 확장성을 제공하는 로깅 프레임워크를 선택하는 것이 중요합니다.

다음은 Ruby on Rails 애플리케이션에서 `ActiveSupport::Logger` 클래스를 사용하여 로깅을 구현하는 예시입니다:

```
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)
```

워크플로우 구성 요소와 AI 의사결정 지점 전반에 걸쳐 로깅 구문을 전략적으로 배치함으로써, 개발자들은 디버깅, 감사 및 분석을 위한 중요한 정보를 수집할 수 있습니다.

모니터링 및 로깅의 이점

지능형 워크플로우 오케스트레이션에서 모니터링과 로깅을 구현하면 다음과 같은 여러 이점이 있습니다:

- 1. 디버깅 및 문제 해결:** 상세한 로그와 모니터링 데이터는 개발자가 문제를 신속하게 식별하고 진단하는 데 도움이 됩니다. 이를 통해 워크플로우 실행 흐름, 구성 요소 간 상호 작용, 발생한 오류나 예외 상황에 대한 통찰력을 제공합니다.
- 2. 성능 최적화:** 성능 지표를 모니터링하면 개발자가 병목 현상을 식별하고 워크플로우 구성 요소를 최적화하여 더 나은 효율성을 달성할 수 있습니다. 실행 시간, 리소스 사용량 및 기타 지표를 분석함으로써 개발자는 시스템의 전반적인 성능을 개선하기 위한 정보에 기반한 결정을 내릴 수 있습니다.
- 3. 감사 및 규정 준수:** 주요 이벤트와 의사결정을 로깅하면 규제 준수와 책임성을 위한 감사 추적이 가능합니다. 이를 통해 조직은 AI 구성 요소가 취한 조치를 추적하고 검증하며 비즈니스 규칙과 법적 요구사항을 준수하도록 보장할 수 있습니다.
- 4. 지속적 개선:** 모니터링 및 로깅 데이터는 지능형 워크플로우의 지속적인 개선을 위한 귀중한 입력 자료가 됩니다. 과거 데이터를 분석하고, 패턴을 식별하며, AI 의사결정의 효과성을 측정함으로써 개발자는 워크플로우 오케스트레이션 로직을 반복적으로 개선하고 향상시킬 수 있습니다.

고려사항 및 모범 사례

지능형 워크플로우 오케스트레이션에서 모니터링과 로깅을 구현할 때 다음과 같은 모범 사례를 고려하십시오:

- 1. 명확한 모니터링 지표 정의:** 워크플로우의 특정 요구사항에 기반하여 모니터링해야 할 주요 지표와 이벤트를 식별하십시오. 시스템의 성능, 상태 및 동작에 대한 의미 있는 통찰력을 제공하는 지표에 집중하십시오.

2. 세분화된 로깅 구현: 워크플로우 구성 요소와 AI 의사결정 지점 내의 적절한 위치에 로깅 구문을 배치하십시오. 입력 매개변수, 출력 결과 및 생성된 중간 데이터와 같은 관련 컨텍스트 정보를 캡처하십시오.
3. 구조화된 로깅 사용: 로그 데이터의 쉬운 구문 분석과 분석을 위해 구조화된 로깅 형식을 채택하십시오. 구조화된 로깅을 통해 로그 항목의 검색성, 필터링 및 집계가 향상됩니다.
4. 로그 보존 및 순환 관리: 로그 파일의 저장소와 수명 주기를 관리하기 위한 로그 보존 및 순환 정책을 구현하십시오. 법적 요구사항, 저장소 제약 조건 및 분석 요구사항에 따라 적절한 보존 기간을 결정하십시오. 가능한 경우 [Papertrail](#)과 같은 제3자 서비스로 로깅을 오프로드하십시오.
5. 민감한 정보 보호: 개인 식별 정보(PII) 또는 기밀 비즈니스 데이터와 같은 민감한 정보를 로깅할 때 주의하십시오. 로그 파일의 민감한 정보를 보호하기 위해 데이터 마스킹이나 암호화와 같은 적절한 보안 조치를 구현하십시오.
6. 모니터링 및 경고 도구와의 통합: 모니터링 및 로깅 데이터의 수집, 분석 및 시각화를 중앙화하기 위해 모니터링 및 경고 도구를 활용하십시오. 이러한 도구는 실시간 통찰력을 제공하고, 사전 정의된 임계값에 기반하여 경고를 생성하며, 사전 예방적인 문제 감지 및 해결을 용이하게 합니다. 이러한 도구 중 제가 가장 선호하는 것은 [Datadog](#)입니다.

포괄적인 모니터링 및 로깅 메커니즘을 구현함으로써 개발자는 지능형 워크플로우의 동작과 성능에 대한 귀중한 통찰력을 얻을 수 있습니다. 이러한 통찰력을 통해 AI 기반 워크플로우 오케스트레이션 시스템의 효과적인 디버깅, 최적화 및 지속적인 개선이 가능해집니다.

확장성 및 성능 고려사항

확장성과 성능은 지능형 워크플로우 오케스트레이션 시스템을 설계하고 구현할 때 고려해야 할 중요한 측면입니다. 동시 워크플로우의 볼륨과 AI 기반 구성 요소의 복잡성이 증가함에 따라, 시스템이 작업량을 효율적으로 처리하고

증가하는 수요를 충족하기 위해 원활하게 확장할 수 있도록 보장하는 것이 필수적입니다.

대량의 동시 워크플로우 처리

지능형 워크플로우 오케스트레이션 시스템은 종종 많은 수의 동시 워크플로우를 처리해야 합니다. 확장성을 보장하기 위해 다음과 같은 전략을 고려하십시오:

- 비동기 처리:** 워크플로우 구성 요소의 실행을 분리하기 위해 비동기 처리 메커니즘을 구현하십시오. 이를 통해 시스템은 각 구성 요소가 완료되기를 기다리거나 차단하지 않고 여러 워크플로우를 동시에 처리할 수 있습니다. 비동기 처리는 메시지 큐, 이벤트 기반 아키텍처 또는 Sidekiq과 같은 백그라운드 작업 처리 프레임워크를 사용하여 구현할 수 있습니다.
- 분산 아키텍처:** AWS Lambda와 같은 서버리스 구성 요소를 사용하거나 메인 애플리케이션 서버와 함께 여러 노드나 서버에 작업량을 분산하도록 시스템 아키텍처를 설계하십시오. 이를 통해 증가된 워크플로우 볼륨을 처리하기 위해 추가 노드를 추가할 수 있는 수평적 확장성이 가능해집니다.
- 병렬 실행:** 워크플로우 내에서 병렬 실행 기회를 식별하십시오. 일부 워크플로우 구성 요소는 서로 독립적일 수 있으며 동시에 실행될 수 있습니다. 멀티스레딩이나 분산 작업 큐와 같은 병렬 처리 기술을 활용함으로써 시스템은 리소스 활용을 최적화하고 전체 워크플로우 실행 시간을 단축할 수 있습니다.

AI 기반 컴포넌트의 성능 최적화

AI 기반 컴포넌트(예: 머신러닝 모델이나 자연어 처리 엔진)는 계산 집약적이며 워크플로우 오케스트레이션 시스템의 전반적인 성능에 영향을 미칠 수 있습니다. AI 컴포넌트의 성능을 최적화하기 위해 다음과 같은 기법들을 고려해보세요:

- 캐싱:** AI 처리가 순수하게 생성적이며 실시간 정보 조회나 외부 통합 없이 채팅 완성을 생성하는 경우, 자주 액세스하거나 계산 비용이 많이 드는 작업의 결과를 저장하고 재사용하기 위한 캐싱 메커니즘을 고려해볼 수 있습니다.
- 모델 최적화:** 워크플로우 컴포넌트에서 AI 모델을 사용하는 방식을 지속적으로 최적화하세요. 여기에는 프롬프트 디스틸레이션과 같은 기법이 포함될 수 있으며, 새로운 모델이 출시될 때마다 테스트하는 것일 수도 있습니다.
- 배치 처리:** GPT-4 클래스 모델을 사용하는 경우, 데이터 포인트나 요청을 개별적으로 처리하는 대신 단일 배치로 처리하는 배치 처리 기법을 활용할 수 있습니다. 데이터를 배치로 처리함으로써 시스템은 리소스 활용을 최적화하고 반복적인 모델 요청의 오버헤드를 줄일 수 있습니다.

성능 모니터링 및 프로파일링

지능형 워크플로우 오케스트레이션 시스템의 성능 병목 현상을 식별하고 확장성을 최적화하기 위해서는 모니터링 및 프로파일링 메커니즘을 구현하는 것이 중요합니다. 다음과 같은 접근 방식을 고려해보세요:

- 성능 메트릭:** 응답 시간, 처리량, 리소스 활용도, 지역 시간과 같은 주요 성능 메트릭을 정의하고 추적하세요. 이러한 메트릭은 시스템 성능에 대한 통찰력을 제공하고 최적화가 필요한 영역을 식별하는 데 도움이 됩니다. 인기 있는 AI 모델 애그리게이터인 [OpenRouter](#)는 각 API 응답에 Host¹와 Speed² 메트릭을 포함하여 이러한 주요 메트릭을 쉽게 추적할 수 있게 합니다.
- 프로파일링 도구:** 개별 워크플로우 컴포넌트와 AI 작업의 성능을 분석하기 위해 프로파일링 도구를 활용하세요. 프로파일링 도구는 성능 핫스팟, 비효율적인 코드 경로, 또는 리소스 집약적인 작업을 식별하는 데 도움이 됩니다. 인기 있는 프로파일링 도구로는 New Relic, Scout, 또는 프로그래밍 언어나 프레임워크에서 제공하는 내장 프로파일러가 있습니다.

¹Host는 모델 호스트로부터 스트리밍된 생성의 첫 번째 바이트를 받는 데 걸린 시간, 즉 “첫 바이트까지의 시간”입니다.

²Speed는 완성 토큰 수를 전체 생성 시간으로 나눈 값입니다. 스트리밍되지 않은 요청의 경우 지역 시간이 생성 시간의 일부로 간주됩니다.

3. 부하 테스트: 다양한 수준의 동시 작업 부하에서 시스템의 성능을 평가하기 위해 부하 테스트를 수행하세요. 부하 테스트는 시스템의 확장성 한계를 식별하고, 성능 저하를 감지하며, 성능을 저하시키지 않고 예상되는 트래픽을 처리할 수 있는지 확인하는 데 도움이 됩니다.

4. 지속적 모니터링: 성능 문제와 병목 현상을 사전에 감지하기 위해 지속적인 모니터링 및 경고 메커니즘을 구현하세요. 주요 성능 지표(KPI)를 추적하고 미리 정의된 임계값이 초과될 때 알림을 받을 수 있도록 모니터링 대시보드와 경고를 설정하세요. 이를 통해 성능 문제를 신속하게 식별하고 해결할 수 있습니다.

확장 전략

지능형 워크플로우 오케스트레이션 시스템의 증가하는 작업 부하를 처리하고 확장성을 보장하기 위해 다음과 같은 확장 전략을 고려하세요:

1. 수직 확장: 수직 확장은 더 높은 작업 부하를 처리하기 위해 개별 노드나 서버의 리소스(예: CPU, 메모리)를 증가시키는 것을 포함합니다. 이 접근 방식은 시스템이 복잡한 워크플로우나 AI 작업을 처리하기 위해 더 많은 처리 능력이나 메모리를 필요로 할 때 적합합니다.

2. 수평 확장: 수평 확장은 작업 부하를 분산시키기 위해 더 많은 노드나 서버를 시스템에 추가하는 것을 포함합니다. 이 접근 방식은 시스템이 많은 수의 동시 워크플로우를 처리해야 하거나 작업 부하를 여러 노드에 쉽게 분산시킬 수 있을 때 효과적입니다. 수평 확장은 분산 아키텍처와 트래픽의 균등한 분배를 보장하기 위한 부하 분산 메커니즘을 필요로 합니다.

3. 자동 확장: 작업 부하 수요에 따라 노드나 리소스의 수를 자동으로 조정하는 자동 확장 메커니즘을 구현하세요. 자동 확장을 통해 시스템은 들어오는 트래픽에 따라 동적으로 확장하거나 축소할 수 있어 최적의 리소스 활용과 비용 효율성을 보장합니다. Amazon Web Services (AWS)나 Google Cloud Platform (GCP)과 같은 클라우드 플랫폼은 지능형 워크플로우 오케스트레이션 시스템에 활용할 수 있는 자동 확장 기능을 제공합니다.

성능 최적화 기법

확장 전략 외에도 지능형 워크플로우 오케스트레이션 시스템의 효율성을 향상시키기 위해 다음과 같은 성능 최적화 기법을 고려하세요:

- 효율적인 데이터 저장 및 검색:** 워크플로우 컴포넌트가 사용하는 데이터 저장 및 검색 메커니즘을 최적화하세요. 효율적인 데이터베이스 인덱싱, 쿼리 최적화 기법, 데이터 캐싱을 사용하여 데이터 집약적인 작업의 지연 시간을 최소화하고 성능을 향상시키세요.
- 비동기 I/O:** 시스템의 응답성을 향상시키고 블로킹을 방지하기 위해 비동기 I/O 작업을 활용하십시오. 비동기 I/O를 통해 시스템은 I/O 작업이 완료되기를 기다리지 않고 여러 요청을 동시에 처리할 수 있어 리소스 활용을 최대화할 수 있습니다.
- 효율적인 직렬화와 역직렬화:** 워크플로우 구성 요소 간의 데이터 교환에 사용되는 직렬화 및 역직렬화 프로세스를 최적화하십시오. Protocol Buffers 또는 MessagePack과 같은 효율적인 직렬화 형식을 사용하여 데이터 직렬화의 오버헤드를 줄이고 구성 요소 간 통신의 성능을 향상시키십시오.



Ruby 기반 애플리케이션의 경우, Universal ID의 사용을 고려해 보십시오. Universal ID는 MessagePack과 Brotli를 모두 활용합니다 (속도와 최고 수준의 데이터 압축을 위해 구축된 조합). 이러한 라이브러리들을 결합하면 Protocol Buffers에 비해 최대 30% 더 빠르며 압축률은 2-5% 이내입니다.

- 압축 및 인코딩:** 워크플로우 구성 요소 간에 전송되는 데이터의 크기를 줄이기 위해 압축 및 인코딩 기술을 적용하십시오. gzip 또는 Brotli와 같은 압축 알고리즘을 사용하면 네트워크 대역폭 사용량을 크게 줄이고 시스템의 전반적인 성능을 향상시킬 수 있습니다.

지능형 워크플로우 오케스트레이션 시스템의 설계 및 구현 과정에서 확장성과 성능 측면을 고려함으로써, 시스템이 높은 수준의 동시 워크플로우를 처리하고,

AI 기반 구성 요소의 성능을 최적화하며, 증가하는 요구 사항을 충족하기 위해 원활하게 확장될 수 있도록 보장할 수 있습니다. 지속적인 모니터링, 프로파일링 및 최적화 노력은 워크로드와 복잡성이 시간이 지남에 따라 증가할 때 시스템의 성능과 응답성을 유지하는 데 필수적입니다.

워크플로우의 테스트 및 검증

테스트와 검증은 지능형 워크플로우 오케스트레이션 시스템을 개발하고 유지하는 데 있어 중요한 측면입니다. AI 기반 워크플로우의 복잡한 특성을 고려할 때, 각 구성 요소가 예상대로 작동하고, 전체 워크플로우가 올바르게 동작하며, AI 결정이 정확하고 신뢰할 수 있도록 보장하는 것이 필수적입니다. 이 섹션에서는 지능형 워크플로우를 테스트하고 검증하기 위한 다양한 기술과 고려사항을 살펴보겠습니다.

워크플로우 구성 요소의 단위 테스트

단위 테스트는 개별 워크플로우 구성 요소를 격리된 환경에서 테스트하여 그 정확성과 견고성을 검증하는 것입니다. AI 기반 워크플로우 구성 요소를 단위 테스트할 때는 다음 사항을 고려하십시오:

- 입력 유효성 검사:** 구성 요소가 유효한 데이터와 유효하지 않은 데이터를 포함한 다양한 유형의 입력을 처리하는 능력을 테스트하십시오. 구성 요소가 엣지 케이스를 우아하게 처리하고 적절한 오류 메시지나 예외를 제공하는지 확인하십시오.
- 출력 검증:** 주어진 입력 세트에 대해 구성 요소가 예상된 출력을 생성하는지 확인하십시오. 실제 출력을 예상 결과와 비교하여 정확성을 보장하십시오.
- 오류 처리:** 잘못된 입력, 리소스 사용 불가, 또는 예기치 않은 예외와 같은 다양한 오류 시나리오를 시뮬레이션하여 구성 요소의 오류 처리 메커니즘을 테스트하십시오. 구성 요소가 오류를 적절하게 포착하고 처리하는지 확인하십시오.

4. 경계 조건: 빈 입력, 최대 입력 크기 또는 극단적인 값과 같은 경계 조건에서 구성 요소의 동작을 테스트하십시오. 구성 요소가 충돌하거나 잘못된 결과를 생성하지 않고 이러한 조건을 우아하게 처리하는지 확인하십시오.

다음은 RSpec 테스트 프레임워크를 사용한 Ruby의 워크플로우 구성 요소에 대한 단위 테스트 예시입니다:

```
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
```

이 예시에서 OrderValidator 컴포넌트는 두 가지 테스트 케이스를 사용하여 테스트됩니다: 하나는 유효한 주문에 대한 것이고 다른 하나는 유효하지 않은 주문에 대한 것입니다. 이 테스트 케이스들은 주문의 유효성에 따라 validate 메서드가 예상된 불리언 값을 반환하는지 확인합니다.

워크플로우 상호작용 통합 테스트

통합 테스트는 서로 다른 워크플로우 컴포넌트 간의 상호작용과 데이터 흐름을 검증하는 데 중점을 둡니다. 이는 컴포넌트들이 원활하게 함께 작동하고 예상된 결과를 생성하는지 확인합니다. 지능형 워크플로우의 통합 테스트를 수행할 때는 다음 사항들을 고려하세요:

- 컴포넌트 상호작용:** 워크플로우 컴포넌트 간의 통신과 데이터 교환을 테스트합니다. 한 컴포넌트의 출력이 워크플로우의 다음 컴포넌트에 올바르게 입력으로 전달되는지 확인합니다.
- 데이터 일관성:** 데이터가 워크플로우를 통해 흐르면서 일관성과 정확성을 유지하는지 확인합니다. 데이터 변환, 계산, 집계가 올바르게 수행되는지 검증합니다.
- 예외 전파:** 예외와 오류가 워크플로우 컴포넌트 전반에 걸쳐 어떻게 전파되고 처리되는지 테스트합니다. 예외가 적절하게 포착되고, 기록되며, 워크플로우 중단을 방지하도록 올바르게 처리되는지 확인합니다.
- 비동기 동작:** 워크플로우가 비동기 컴포넌트나 병렬 실행을 포함하는 경우, 조정 및 동기화 메커니즘을 테스트합니다. 동시 실행 및 비동기 시나리오에서 워크플로우가 올바르게 동작하는지 확인합니다.

다음은 RSpec 테스팅 프레임워크를 사용한 Ruby 워크플로우의 통합 테스트 예시입니다:

```
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
```

이 예시에서는 워크플로우 컴포넌트들 간의 상호작용을 검증하여 `OrderProcessingWorkflow`를 테스트합니다. 테스트 케이스는 각 컴포넌트의 동작에 대한 기대치를 설정하고, 워크플로우가 주문을 성공적으로 처리하여 주문 상태를 적절히 업데이트하는지 확인합니다.

AI 의사결정 포인트 테스트

AI 의사결정 포인트를 테스트하는 것은 AI 기반 워크플로우의 정확성과 신뢰성을 보장하는 데 매우 중요합니다. AI 의사결정 포인트를 테스트할 때는 다음 사항들을 고려하세요:

1. 의사결정 정확도: AI 컴포넌트가 입력 데이터와 학습된 모델을 기반으로 정확한 의사결정을 내리는지 확인합니다. AI의 의사결정을 예상 결과나 실측 데이터와 비교합니다.
2. 옛지 케이스: 옛지 케이스와 특이한 시나리오에서 AI 컴포넌트의 동작을 테스트합니다. AI 컴포넌트가 이러한 케이스들을 적절히 처리하고 합리적인 의사결정을 내리는지 확인합니다.
3. 편향성과 공정성: AI 컴포넌트의 잠재적 편향성을 평가하고 공정하고 편향되지 않은 의사결정을 내리는지 확인합니다. 다양한 입력 데이터로 컴포넌트를 테스트하고 차별적인 패턴이 있는지 결과를 분석합니다.
4. 설명 가능성: AI 컴포넌트가 의사결정에 대한 설명이나 근거를 제공하는 경우, 설명의 정확성과 명확성을 검증합니다. 설명이 기저의 의사결정 프로세스와 일치하는지 확인합니다.

다음은 RSpec 테스팅 프레임워크를 사용하여 Ruby로 AI 의사결정 포인트를 테스트하는 예시입니다:

```
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
```

이 예시에서는 FraudDetector AI 컴포넌트를 두 가지 테스트 케이스로 검증합니다: 하나는 사기성 거래에 대한 것이고 다른 하나는 정상적인 거래에 대한 것입니다. 이 테스트 케이스들은 detect_fraud 메서드가 거래의 특성에 따라 예상되는 불리언 값을 반환하는지 확인합니다.

엔드 투 엔드 테스트

엔드 투 엔드 테스트는 실제 상황과 사용자 상호작용을 시뮬레이션하여 처음부터 끝까지 전체 워크플로우를 테스트하는 것입니다. 이는 워크플로우가 올바르게 작동하고 원하는 결과를 생성하는지 확인합니다. 지능형 워크플로우에 대한 엔드 투 엔드 테스트를 수행할 때는 다음 사항들을 고려하세요:

- 1. 사용자 시나리오:** 일반적인 사용자 시나리오를 식별하고 이러한 시나리오에서 워크플로우의 동작을 테스트합니다. 워크플로우가 사용자 입력을 올바르게 처리하고, 적절한 결정을 내리며, 예상된 출력을 생성하는지 확인합니다.
- 2. 데이터 유효성 검사:** 워크플로우가 데이터 불일치나 보안 취약점을 방지하기 위해 사용자 입력을 검증하고 정제하는지 확인합니다. 유효한 데이터와 유효하지 않은 데이터를 포함한 다양한 유형의 입력 데이터로 워크플로우를 테스트합니다.

3. 오류 복구: 워크플로우의 오류 및 예외 상황 복구 능력을 테스트합니다. 오류 시나리오를 시뮬레이션하고 워크플로우가 이를 적절히 처리하고, 오류를 기록하며, 적절한 복구 조치를 취하는지 확인합니다.

4. 성능 및 확장성: 다양한 부하 조건에서 워크플로우의 성능과 확장성을 평가합니다. 대량의 동시 요청으로 워크플로우를 테스트하고 응답 시간, 리소스 사용량 및 전반적인 시스템 안정성을 측정합니다.

다음은 Ruby에서 RSpec 테스팅 프레임워크와 사용자 상호작용을 시뮬레이션하기 위한 Capybara 라이브러리를 사용한 워크플로우의 엔드 투 엔드 테스트 예시입니다:

```
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
```

이 예시에서 엔드투엔드 테스트는 사용자가 웹 인터페이스를 통해 주문하는 과정을 시뮬레이션합니다. 필요한 양식 필드를 작성하고, 주문을 제출한 다음, 주문이 성공적으로 처리되어 적절한 확인 메시지가 표시되고 데이터베이스의 주문 상태가 업데이트되는 것을 확인합니다.

지속적 통합 및 배포

지능형 워크플로우의 신뢰성과 유지보수성을 보장하기 위해, 테스트와 검증을 지속적 통합 및 배포(CI/CD) 파이프라인에 통합하는 것이 권장됩니다. 이를 통해 워크플로우 변경사항이 프로덕션에 배포되기 전에 자동화된 테스트와 검증이 가능해집니다. 다음과 같은 사례들을 고려해보세요:

- 1. 자동화된 테스트 실행:** 워크플로우 코드베이스에 변경사항이 있을 때마다 테스트 스위트를 자동으로 실행하도록 CI/CD 파이프라인을 구성합니다. 이를 통해 개발 프로세스 초기 단계에서 회귀나 실패를 감지할 수 있습니다.
- 2. 테스트 커버리지 모니터링:** 워크플로우 구성 요소와 AI 의사결정 지점의 테스트 커버리지를 측정하고 모니터링합니다. 중요한 경로와 시나리오가 철저히 테스트되도록 높은 테스트 커버리지를 목표로 합니다.
- 3. 지속적 피드백:** 테스트 결과와 코드 품질 메트릭을 개발 워크플로우에 통합합니다. CI/CD 프로세스 중에 감지된 테스트 상태, 코드 품질 및 문제에 대해 개발자들에게 지속적인 피드백을 제공합니다.
- 4. 스테이징 환경:** 프로덕션 환경과 최대한 유사한 스테이징 환경에 워크플로우를 배포합니다. 스테이징 환경에서 추가 테스트와 검증을 수행하여 인프라, 구성 또는 데이터 통합과 관련된 문제를 포착합니다.
- 5. 롤백 메커니즘:** 배포 실패나 프로덕션에서 발견된 심각한 문제에 대비하여 롤백 메커니즘을 구현합니다. 다운타임과 사용자 영향을 최소화하기 위해 워크플로우를 이전의 안정적인 버전으로 신속하게 되돌릴 수 있도록 합니다.

지능형 워크플로우의 개발 수명 주기 전반에 걸쳐 테스트와 검증을 통합함으로써, 조직은 AI 기반 시스템의 신뢰성, 정확성 및 유지보수성을 보장할 수 있습니다. 정기적인 테스트와 검증은 버그를 포착하고, 회귀를 방지하며, 워크플로우의 동작과 결과에 대한 신뢰를 구축하는 데 도움이 됩니다.

Part 2: 패턴들

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

프롬프트 엔지니어링

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

사고 연쇄

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

콘텐츠 생성

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

구조화된 개체 생성

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

LLM 에이전트 지도

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

이점과 고려사항

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

모드 전환

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

사용 시기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

역할 할당

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

사용 시기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

프롬프트 객체

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

프롬프트 템플릿

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

이점과 고려사항

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

사용 시기:

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

구조화된 입출력

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

구조화된 입출력의 확장

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

이점과 고려사항

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

프롬프트 체이닝

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

사용 시기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시: Olympia의 온보딩

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

프롬프트 리라이터

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

응답 펜싱

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

이점과 고려사항

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

오류 처리

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

쿼리 분석기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

구현

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

품사 태깅(POS)과 개체명 인식(NER)

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

의도 분류

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

키워드 추출

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

이점

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

쿼리 재작성기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

이점

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

Ventriloquist

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

사용 시기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

이산 컴포넌트

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

조건자

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

사용 시기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

API 퍼사드

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

주요 이점

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

사용 시기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

인증 및 권한 부여

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

요청 처리

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

응답 형식 지정

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

오류 처리 및 예외 상황

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

확장성 및 성능 고려사항

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

다른 디자인 패턴과의 비교

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

결과 해석기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

사용 시기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

가상 머신

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

사용 시기

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

마법 같은 원리

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

명세와 테스트

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

동작 명세화

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

테스트 케이스 작성

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시: 번역기 컴포넌트 테스트

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

HTTP 상호작용 재생

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

휴먼 인 더 루프 (HITL)

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

상위 수준 패턴

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

하이브리드 인텔리전스

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

적응형 응답

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

인간-AI 역할 전환

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

에스컬레이션

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

주요 이점

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

실제 응용 사례: 의료 분야

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

피드백 루프

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

응용 및 예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

인간 피드백 통합의 고급 기법

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

수동적 정보 방사

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

맥락적 정보 표시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

선제적 알림

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

설명적 통찰

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

대화형 탐색

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

주요 이점

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

응용 및 예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

협력적 의사결정 (CDM)

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

지속적 학습

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

응용 및 예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

윤리적 고려사항

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

AI 리스크 완화에서의 인간 참여 시스템의 역할

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

기술적 발전과 미래 전망

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

HITL 시스템의 과제와 한계

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

지능형 오류 처리

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

전통적인 오류 처리 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

맥락적 오류 진단

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

맥락적 오류 진단을 위한 프롬프트 엔지니어링

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

맥락적 오류 진단을 위한 검색 증강 생성

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

지능형 오류 보고

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예측적 오류 방지

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

스마트 오류 복구

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

개인화된 오류 커뮤니케이션

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

적응형 오류 처리 워크플로우

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

품질 관리

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

Eval

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

문제

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

해결책

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

고려사항

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

골든 레퍼런스의 이해

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

참조 없는 평가의 작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

안전장치

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

문제

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

해결책

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

작동 방식

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

예시

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에서 구매할 수 있습니다.

고려사항

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

가드레일과 평가: 동전의 양면

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

가드레일과 참조 없는 평가의 호환성

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

이중 목적 가드레일과 평가 구현

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

용어집

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

용어집

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

A

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

B

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

C

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

D

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

E

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

F

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

G

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

H

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

I

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

J

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

K

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

L

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

M

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

N

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

O

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

P

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

Q

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

R

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

S

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

T

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

U

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

V

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

W

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

Z

이 내용은 샘플 책에 포함되어 있지 않습니다. 책은 Leanpub에서 <http://leanpub.com/patterns-of-application-development-using-ai-ko>에 서 구매할 수 있습니다.

Index

ACID 속성, 104

AI, 59, 67, 94, 124, 130, 139, 146,
204

applications, 134

대화형, 6, 28, 205

모델, 83, 94, 151, 153, 155, 204

복합 시스템, 27, 28, 31

애플리케이션, 120, 145, 159

의사결정 포인트, 250

AI 작업자의 체이닝, 106

Alpaca, 12

Altman, Sam, 16

Amazon Web Services, 245

Anthropic, 20, 35, 67, 124, 133

API, 65, 118, 150

audit logging, 102

BERT, 12, 22

Byte Pair Encoding (BPE), 13

C (프로그래밍 언어), 111

Capybara 라이브러리, 253

Chain of Thought (CoT), 135

ChatGPT, 27, 47

classification, 115

Claude, 7, 39, 71

Claude 3, 45, 121, 124, 130, 133

Claude 3 Opus, 68

Claude v1, 15

Claude v2, 15

Cohere (LLM 제공업체), 20, 22

context

infinitely long inputs, 14

window, 14

data

persistence, 104

preparation, 104

Databricks 직원들, 46

Datadog, 242

document clustering, 115

Dohan, et al., 39

ELK 스택, 105

ensembles, 112

errors

handling, 102

지능형 오류 처리, 139

event-driven architecture, 103

F#, 87

Facebook, 22

finalize method, 152

finalize 메서드, 154, 155

FitAI, 205

- function
 - call history, 152
- Gemma 7B, 10
- GitLab, 87
- Google,
 - API, 56, 59
 - Cloud AI Platform, 22
 - Cloud Platform, 245
 - Gemini, 19
 - Gemini 1.5 Pro, 12, 15, 17
 - PaLM (Pathways Language Model), 16, 22
 - T5, 12
- GPT-3, 12, 15
- GPT-4, 6, 12, 15, 16, 19, 28, 39, 45, 56, 99, 112, 115, 122, 129, 197, 198, 244
- Graham, Paul, 17
- GraphQL, 103
- Groq, 23, 115
- gzip, 246
- Hohpe, Gregor, 99
- Honeybadger, 88
- HTTP, 146
- JSON (JavaScript Object Notation), 126, 127, 130, 143, 162
- JSON(JavaScript Object Notation), 121
- K-means, 116
- language
 - models, 38, 59
- Large Language Model (LLM), 14, 16, 115
- Latent Dirichlet Allocation, 116
- Llama, 12
 - Llama 2–70B, 45
 - Llama 3 70B, 10
 - Llama 3 8B, 10
 - LLM 통합, 182
- Louvre, 38
- majority voting, 112
- Managed Streaming for Apache Kafka, 37
- Memorial Sloan Kettering Cancer Center, 37
- Meta, 22
- Metropolitan Museum of Art, 38
- Mistral, 23
 - 7B, 10
 - 7B Instruct, 15, 198
- Mixtral
 - 8x22B, 10
 - 8x7B, 50
- Naive Bayes, 115
- natural language
 - Natural Language Processing (NLP), 115
- New Relic, 244
- Ollama, 23

- Olympia, 30, 56, 124, 139, 147
Olympia의 지식 베이스, 86
OpenAI, 3, 20, 35, 67
OpenRouter, 25, 26, 147, 244
OPT model, 22

Perplexity (제공업체), 10
Process Manager, 102
prompts
 engineering, 59
publish-subscribe systems, 103
PyTorch, 22

Qwen2 70B, 10

Rails, 188
Railway Oriented Programming
 (ROP), 89
Raix, 221
 라이브러리, 92
RSpec, 248, 249, 253
Ruby, 87, 88, 108, 159, 253
Ruby on Rails, 1, 106, 221
Rudall, Alex, 21
Rust (Programming Language), 87
Rust (프로그래밍 언어), 111

Scout, 244
sentiment analysis, 112
server-sent events (SSE), 146
SQL 인젝션, 64
stream handlers, 146
stream processing, 146, 152
Stripe, 124

Support Vector Machines (SVM), 115

T5, 22
Together.ai, 23
Top-k 샘플링, 43
Top-p (핵) 샘플링, 43
topic identification, 115

Unicode-encodable language, 13

Wall, Larry, 3
Wisper, 89, 101, 147, 155
Wooley, Chad, 87

XML, 130

Yi-34B, 45

가상 비서, 30
감사 및 규정 준수, 241
감성 분석, 15, 113, 141
감정 분석, 96, 106, 107, 109, 130
감정 톤, 141
강제 도구 선택, 127
개념적 및 실용적 도전과제, 192
개발 프레임워크, 144
개인화, 182, 211, 215
 개인화된 마이크로카페, 200
 개인화된 양식, 193
개인화된 상품 추천, 86
검색 강화 생성(RAG), 28
검색 기반 모델, 6
검색 증강 생성 (RAG), 41
검색 증강 생성(RAG), 34, 74, 120

- 결과 해석기, 138
결정론적 동작, 52
경계 조건, 248
경로 좁히기, 34, 35
계정, 85
고객 감정 분석, 95
고객 서비스 챗봇, 30
고객 지원, 29
고성능 완성, 24
공급망
 최적화, 29
공유지의 비극, 185
과거 패턴, 217
교육용 애플리케이션, 29
교통 관리, 29
구문 오류, 127
구조화된 IO, 199
구조화된 데이터, 129
구조화된 로깅, 242
국제화, 188
그래프 모델, 39
기본 모델, 48

낙관적 잠금, 104
네트워크 연결, 218
다단계 워크플로우, 106
다수의 워커, 162
다중 에이전트
 문제 해결사, 28
다중 워커, 113
대규모 언어 모델 (LLM), 26, 65, 82,
 137, 140, 160, 162, 191,
 197, 203, 224
대규모 언어 모델(LLM), 1, 3, 61, 62,
 70, 72, 105, 118, 120, 130,
 140, 143, 181
 환경, 24
대체 전략, 104
대화
 기록, 153, 155
 루프, 156
데스크톱 컴퓨터, 211
데이터
 데이터 검색, 104
 데이터 동기화, 104
 데이터 유효성 검사, 252
 무결성, 233
 분석, 31, 143
 처리 작업, 120
 처리 파이프라인, 233
 프라이버시, 24, 209
 흐름, 105
데이터베이스, 118
 기반 객체, 100
 잠금 전략, 104
도구 사용, 118, 145
도구 호출, 149
동기부여 전략, 207
동시 워크플로우, 246
동적 UI 생성, 182
동적 도구 선택, 127
동적 작업 라우팅, 215
디버깅, 217
 및 문제 해결, 241

- 및 테스팅, 128
디지털 환경, 187
딕셔너리, 126
로그 보존 및 순환, 242
로컬 개발 환경, 151
롤백 메커니즘, 254
롤플레이 스타일 상호작용, 6
마음 이론, 36
マイ크로서비스 아키텍처, 84
마크다운, 143
마크업 스타일 태깅, 64
매개변수
 매개변수 수, 25
 범위, 10
 영향, 124
맥락
 맥락 기반 의사결정, 217
 맥락 기반 콘텐츠 생성, 181, 192, 193
 맥락 기반 필드 제안, 193
 맥락적 콘텐츠 생성, 185, 186
 윈도우, 217
멀티모달
 모델, 18
 언어 모델, 19
메르쿠리우스(로마 신), 40
메시지팩, 246
명령어 튜닝, 9
 명령어 튜닝 모델, 45
명령줄
 명령줄 인터페이스 (CLI), 23
모니터링
 및 경고, 219
 및 로깅, 105, 240
 지표, 241
모듈성, 83
무상태, 153
문법 규칙, 4
바꿔쓰기, 47
바이트 쌍 인코딩 (BPE), 12
반복 페널티, 45
반복적 개선, 140
반복적 정제, 70
배열, 126
배치 처리, 244
번역, 15, 189
벤트릴로퀴스트, 172
병렬 실행, 243
병목 현상, 217
보험 확인, 96
복잡한 작업, 142
분류, 46
분산 아키텍처, 243
브로틀리, 246
비관적 잠금, 104
비동기 처리, 243
비상 대응 계획, 29
비즈니스 규칙, 213
비지도 학습, 4
사고 연쇄 (CoT), 40
사기 탐지
 시스템, 92
사용성 문제, 209

- 사용자 경험, 187
사용자 생성 콘텐츠, 106
사용자 신뢰, 210
사용자 심리학, 208
사용자 인터페이스 (UI)
 기술, 203
 디자인, 211
 인터페이스, 191, 207
 프레임워크, 207
사용자 테스트와 피드백, 190
상품 추천, 86
생산성, 184
생성형 UI (GenUI), 191, 199, 203,
 207, 210
생성형 사전학습 트랜스포머(GPT), 7,
 61
생태계, 144
서사 구축, 18
선형 회귀, 39
선형대수, 39
설명 가능성, 251
성능
 문제, 245
 최적화, 128, 189, 241
 트레이드오프, 5
세분화 및 타겟팅 전략, 187
세분화된 로깅, 242
소프트웨어 아키텍처, 2
수동 개입, 220
수성(행성), 40
수온(원소), 40
순위 결정기, 32
스마트폰, 211
스테이징 환경, 254
스트리밍 데이터, 148
스트림 처리, 159
 로직, 155
시각적 인터페이스, 203
시스템 지시어, 94, 123
신경망, 3, 6
실험
 프레임워크, 187
양상블, 113
 워커 양상블, 113
애플리케이션 설계와 프레임워크, 191
양자화, 26
언어
 관련 작업, 4
 모델, 66
 언어 감지, 106
에이전틱, 29
엔드 투 엔드 테스트, 252
엔드투엔드 테스팅, 253
엔터프라이즈 애플리케이션 아키텍처,
 34
엔터프라이즈 통합 패턴, 99
엣지 케이스, 52
예외 처리, 218, 220
예측, 5
오류
 복구, 253
 비율, 105
 처리, 104, 138, 247
오픈소스 모델 호스팅 제공업체, 199

- 온도, 48
온라인 소매업체, 199
올림피아, 163
외부 서비스 또는 API, 121
요약, 46
원샷 러닝, 54
위험 요소, 90, 91
위험도 분류, 97
유니버설 ID, 246
유연성과 창의성, 189
윤리
 함의, 192
음성 제어 인터페이스, 30
응답 편성, 172, 199
의료 이력 수집, 96
의사결정
 사용 사례, 129
 지점, 239
 트리, 214
의사결정 능력, 94
의학적 발견, 96
이커머스, 185
이커머스 애플리케이션, 86
인격화, 62
인공지능, 195
일관성
 및 재현성, 128
임상 의사결정 지원, 98
입력
 유효성 검사, 247
 프롬프트, 50
입력 매개변수, 123
자가 치유 데이터, 160, 238
자기회귀 모델링, 39
자동 연속, 156
자동 확장, 245
자연어
 자연어 처리 (NLP), 96
잠재 공간, 36, 38
재시도 메커니즘, 104
적응형 UI, 202
적응형 워크플로우
 적응형 워크플로우 구성, 217
전역 인터프리터 잠금 (GIL), 110
점진적 공개, 201
접근성, 210
정보
 검색, 6, 121
 추출, 46
제로샷 러닝, 52, 53
존재 페널티, 44
주요 지표 추적, 238
주요 패턴, 215
증강현실 안경, 211
종류 과정, 70
증상 평가 및 분류, 96
지능형 워크플로우 오케스트레이션,
 220, 244, 246
지능형 콘텐츠 관리자, 225
지속적 통합 및 배포 (CI/CD), 254
 파이프라인, 254
지속적인 위험 모니터링, 98
지식 관리, 29
지식 베이스, 7

- 지연 시간, 25
질의응답 시스템, 7
창의적 글쓰기, 46
창작 글쓰기, 31
챗봇 애플리케이션, 113
처리 시간, 105
처리량, 25
첫 토큰 생성 시간 (TTFT), 25
최소 권한의 원칙, 65
추론, 5
출력 검증, 247
캐싱, 244
커스터마이제이션, 24
컨텍스트
 증강, 41
컴퓨터 과학, 64, 66
콘텐츠
 콘텐츠 분류, 106
 필터링, 24
콘텐츠 기반 필터링, 86
크로스모달 생성, 20
클로드, 71
태블릿, 211
텍스트 정리, 106
토큰, 5, 11
토크나이저, 11
통합 테스트, 249
트랜스포머 아키텍처, 6
트리거 메시지, 99
티켓 배정, 233
파인튜닝, 74
패턴 매칭, 148
편향성
 AI의 편향성과 공정성, 251
폐쇄형 및 개방형 질문 답변, 46
포용적 인터페이스, 192
퓨샷 러닝, 55
퓨샷 프롬프팅, 56
프로세스 관리자, 99
 엔터프라이즈 통합, 221
프로토콜 버퍼, 246
프롬프트
 개선, 62
 설계, 52, 62
 엔지니어링, 36, 41, 50, 53, 61,
 208
 체이닝, 53, 65
 프롬프트 객체, 68
 프롬프트 디스틸레이션, 244
 프롬프트 정제, 72
 프롬프트 종류, 41, 67
 프롬프트 템플릿, 53, 199
피드백
 피드백 루프, 53
하드웨어, 25
하이퍼파라미터, 42
학습 데이터, 38
함수
 이름, 150
 호출, 118
 함수 호출 실패, 129
 함수형 프로그래밍, 86
 합성 데이터 생성, 47

해시, 148

현대적 애플리케이션, 215

협업 필터링, 86

확률적 모델, 39

확장성, 215, 242

회로 차단기 로직, 158

효율성, 214

휴먼 인 더 루프 (HITL), 174