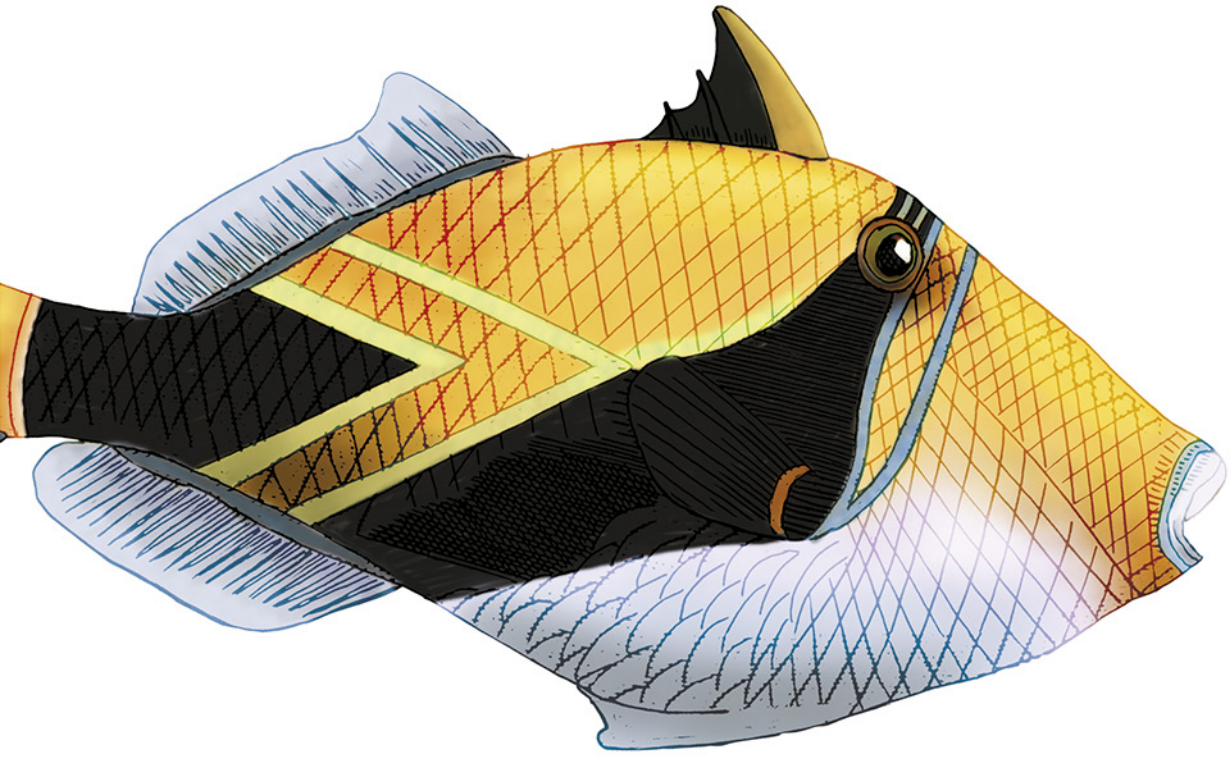
O'REILLY®

Programowanie wspomagane sztuczną inteligencją

Lepsze planowanie, kodowanie, testowanie i wdrażanie



Helion

Tom Taulli

Tytuł oryginału: AI-Assisted Programming: Better Planning, Coding, Testing, and Deployment

Tłumaczenie: Anna Mizerska

ISBN: 978-83-289-2082-8

© 2025 Helion S.A.

Authorized Polish translation of the English edition of *AI-Assisted Programming*
ISBN 9781098164560 © 2024 Tom Taulli.

This translation is published and sold by permission of O'Reilly Media, Inc.,
which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any
form or by any means, electronic or mechanical, including photocopying, recording
or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości
lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione.
Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie
książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie
praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi
bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje
były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich
wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych
lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności
za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/prwssz
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

ROZDZIAŁ 3.

Inżynieria promptów

Inżynieria promptów jest poddziedziną uczenia maszynowego i **przetwarzania języka naturalnego**, która bada, jak umożliwić komputerom rozumienie i interpretowanie ludzkiego języka. Głównym celem jest znalezienie sposobu, jak rozmawiać z **dużymi modelami językowymi** (zaawansowanymi systemami sztucznej inteligencji zaprojektowanymi do przetwarzania i generowania odpowiedzi przypominających ludzką mowę), aby generowały odpowiedź, której szukamy.

Pisanie promptów można porównać z sytuacją, kiedy pytasz kogoś o radę i musisz dać mu trochę kontekstu i jasno określić, czego potrzebujesz. Tak samo jest z modelami LLM. Trzeba starannie sformułować swoje pytanie lub prompt. Czasami może być nawet konieczne dodanie wskazówek lub dodatkowych informacji w pytaniu, aby mieć pewność, że model LLM zrozumie, o co prosisz.

To nie tylko kwestia zadawania pojedynczych pytań. Czasami prowadzi się całą rozmowę z modelem LLM, wymieniając się pytaniami i odpowiedziami, dopóki nie uzyska się tej cennej informacji, której się potrzebuje.

Powiedzmy, że na przykład używasz narzędzia do programowania wspomaganego przez sztuczną inteligencję do tworzenia aplikacji internetowej. Zaczynasz od pytania, jak stworzyć prosty system logowania użytkowników w języku JavaScript. Początkowa odpowiedź może obejmować podstawy, ale potem zdajesz sobie sprawę, że potrzebujesz bardziej zaawansowanych funkcji. Kontynuujesz więc z bardziej szczegółowymi promptami, pytając o włączenie szyfrowania haseł i bezpieczne łączenie się z bazą danych. Każda interakcja ze sztuczną inteligencją doskonali jej odpowiedź, stopniowo kształtując ją tak, aby pasowała do konkretnych potrzeb Twojego projektu.

Pamiętaj, że inżynieria promptów stała się bardzo pożądanym zawodem. Według danych z Willis Towers Watson (*https://oreil.ly/Qy9Zi*) średnie roczne zarobki inżyniera promptów wynoszą około 130 tysięcy dolarów, chociaż ta kwota może być niedoszacowana. Aby przyciągnąć najlepsze talenty, firmy często uatrakcyjniają ofertę kuszącymi pakietami akcji i bonusami.

W tym rozdziale zagłębimy się w świat inżynierii promptów i poznamy pomocne strategie oraz sztuczki.

Kup książkę Poleć książkę

Sztuka i nauka

Inżynieria promptów to mieszanka sztuki i nauki. Musisz wybrać odpowiednie słowa i ton, aby sztuczna inteligencja odpowiedziała tak, jak tego chcesz. Chodzi o kierowanie rozmową w określonym kierunku. Wymaga to pewnej intuicji i kreatywnego podejścia w celu doskonalenia swojego języka, by wydobyć szczegółowe i pełne niuansów odpowiedzi.

Tak, to może być trudne, zwłaszcza dla programistów. Zazwyczaj pisząc kod, postępujesz zgodnie z zestawem reguł, a kod albo działa, albo kompilator informuje Cię, co zrobiłeś źle. Jest to logiczne i przewidywalne.

A inżynieria promptów? To już wygląda nieco inaczej, gdyż jest bardziej dowolna i nieprzewidywalna.

Jednak jest w tym również sporo nauki. Musisz zrozumieć mechanizmy działania modeli sztucznej inteligencji, o których rozmawialiśmy w rozdziale 2. Oprócz kreatywności potrzebujesz dokładności, przewidywalności oraz zdolności do powtarzania wyników. Często oznacza to konieczność eksperymentowania, wypróbowywania różnych promptów, analizowania wyników i wprowadzania poprawek aż do momentu uzyskania odpowiedniej odpowiedzi.

Jeśli chodzi o inżynierię promptów, nie oczekuj magicznych rozwiązań, które będą działać za każdym razem. Owszem jest wiele kursów, filmów i książek, w których twierdzi się, że odkrywają one wszystkie „sekrety" inżynierii promptów. Ale przyjmij je z przymrużeniem oka, bo możesz się rozczarować.

Ponadto świat sztucznej inteligencji i uczenia maszynowego ciągle się zmienia, pojawiają się nowe modele i techniki. Tak więc idea posiadania jednej, ostatecznej techniki dla inżynierii promptów nie wydaje się realna.

Wyzwania

Inżynieria promptów może być frustrująca. Nawet najmniejsza zmiana w sformułowaniu promptu może znacząco wpłynąć na to, co wygeneruje model LLM. Wynika to z zaawansowanej technologii opartej na probabilistycznych ramach.

Oto niektóre wyzwania związane z inżynierią promptów:

Rozwlekłość

Duże modele językowe mogą być „gawędziarzami". Podasz im prompt, a one mogą zacząć mówić bez opamiętania, dając rozległą odpowiedź, podczas gdy Ty chciałeś tylko szybkiej odpowiedzi. Mają tendencję do dodawania mnóstwa pokrewnych pomysłów lub faktów, czyniąc odpowiedź dłuższą, niż to konieczne. Jeśli chcesz, aby model LLM przeszedł od razu do sedna, po prostu poproś go o bycie „zwięzłym".

Brak możliwości użycia z innym modelem

Oznacza to, że prompt, który działa dobrze z jednym modelem LLM, może nie być równie skuteczny z innym. Jeśli więc przechodzisz z ChatGPT na Gemini lub GitHub Copilot, możesz być zmuszony dostosować swoje prompty ze względu na unikatowe szkolenie,

Kup książkę Poleć książkę

projekt i specjalizację każdego modelu LLM. Różne modele są trenowane na różnych zbiorach danych i algorytmach, co prowadzi do odmiennych sposobów rozumienia i interpretacji promptów.

Wrażliwość na długość

Modele LLM mogą czuć się przytłoczone długimi promptami i zacząć pomijać lub źle interpretować części Twoich danych wejściowych. To tak, jakby koncentracja modelu LLM osłabła, a jego odpowiedzi stały się nieco chaotyczne. Dlatego powinieneś unikać podawania zbyt szczegółowych wymagań w promptach. Staraj się, by Twoje prompty były nie dłuższe niż jedna strona.

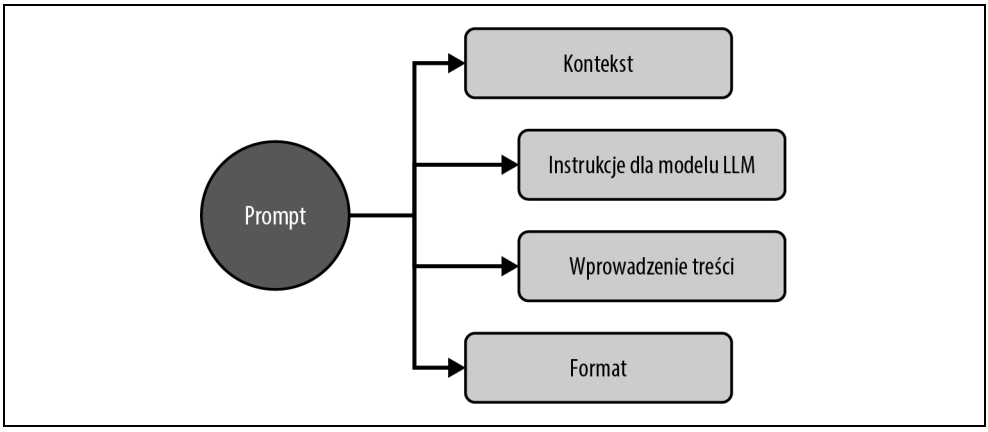
Niejasność

Jeśli Twój prompt jest niejasny, model LLM może się pogubić i podać odpowiedzi, które są zupełnie nie na temat lub po prostu zmyślone. Kluczowa jest przejrzystość.

Mimo wymienionych właśnie wyzwań istnieją sposoby na poprawę wyników, które omówimy w dalszej części tego rozdziału.

Prompt

Można przyjąć, że prompt składa się z czterech głównych elementów pokazanych na rysunku 3.1.



Rysunek 3.1. Prompt ma cztery główne części składowe

Po pierwsze, **kontekst** określa rolę lub personę, jaką model LLM ma przyjąć, udzielając odpowiedzi. Następnie są **instrukcje**, takie jak streszczanie, tłumaczenie lub klasyfikowanie. Kolejnym elementem jest **wprowadzenie treści**, jeśli chcesz, aby model LLM przetworzył informacje w celu przygotowania lepszej odpowiedzi. Na koniec możesz określić format danych wyjściowych.

Pamiętaj, że nie musisz zawierać w prompcie wszystkich tych elementów. W rzeczywistości może wystarczyć tylko jeden, aby uzyskać dobrą odpowiedź. Jednak lepiej — to ogólna zasada — dostarczyć modelowi LLM bardziej konkretnych szczegółów.

Przyjrzyjmy się teraz bliżej każdemu z tych elementów.

Kup książkę Poleć książkę

Kontekst

Często prompt rozpoczyna się od zdania lub dwóch, które dostarczają kontekstu. Często określa się rolę lub personę, którą sztuczna inteligencja ma przyjąć podczas udzielania odpowiedzi. To prowadzi do odpowiedzi, które są nie tylko dokładniejsze, ale także kontekstowo odpowiednie, zapewniając bardziej znaczący wynik.

Na przykład jeśli chcesz usunąć błędy w danym fragmencie kodu, możesz użyć takiego kontekstu:

Prompt:

```
Jesteś doświadczonym inżynierem oprogramowania specjalizującym się w debugowaniu
aplikacji napisanej w języku Java.
```

Albo załóżmy, że chcesz dowiedzieć się więcej o technikach optymalizacji dla konkretnego algorytmu. Możesz określić kontekst w następujący sposób:

Prompt:

```
Jesteś starszym programistą z ekspercką wiedzą na temat optymalizacji algorytmów.
```

Dodanie kontekstu pomaga modelowi LLM podejść do Twojego promptu z odpowiednim nastawieniem.

Instrukcje

Twój prompt powinien zawierać przynajmniej jedną jasną instrukcję. Nic nie stoi na przeszkodzie, aby dodać więcej instrukcji, ale musisz być ostrożny. Przeładowanie promptu wieloma zapytaniami może zdezorientować model LLM, a co za tym idzie, utrudnić uzyskanie oczekiwanej odpowiedzi.

Przyjrzyjmy się bliżej, dlaczego tak się dzieje. Gdy masz wiele instrukcji, mogą stać się one nieco niejasne. Jeśli nie są precyzyjne albo jeśli wydają się nawzajem sprzeczne, model LLM może się pogubić, na której instrukcji się skupić lub jak je wszystkie zrównoważyć.

Poza tym większa liczba instrukcji oznacza, że model LLM ma więcej elementów do przetworzenia. Musi zrozumieć każdą część Twojego promptu, a następnie połączyć wszystkie części w spójną odpowiedź. To dużo „mentalnej gimnastyki", co czasem może prowadzić do błędów lub odpowiedzi, które są nietrafione.

I nie zapominaj, że duże modele językowe przetwarzają instrukcje po kolei, jedna po drugiej. A więc sposób, w jaki uporządkujesz swoje zapytania, może wpłynąć na to, jak zostaną zinterpretowane i jaką odpowiedź otrzymasz.

Gdy weźmie się to wszystko pod uwagę, dobrym pomysłem jest upraszczanie promptów. Zamiast rzucać modelowi LLM całą listę pytań naraz, spróbuj podzielić je na serię mniejszych promptów. To jak prowadzenie żywego dialogu zamiast wygłaszania monologu.

Istnieje również wiele rodzajów instrukcji dla promptów. W kolejnych sekcjach omówimy niektóre spośród głównych instrukcji używanych w programowaniu.

Kup książkę Poleć książkę

Streszczanie

Streszczanie może skrócić dłuższy tekst do krótszej wersji przy zachowaniu głównych idei i punktów. Jest to przydatne do szybkiego zrozumienia długich dokumentów. Dla programisty streszczanie może być szczególnie przydatnym narzędziem w sytuacjach wymienionych w tabeli 3.1.

Tabela 3.1. Streszczone prompty dla zadań z programowania

Przypadek użycia	Opis	Przykładowy prompt
Dokumentacja kodu	Zapewnia zwięzły przegląd obszernej dokumentacji, podkreślając kluczowe funkcjonalności, zależności i struktury.	„Streść główne punkty poniższej dokumentacji, aby zapewnić szybki przegląd bazy kodu".
Raportowanie błędów	Szybko identyfikuje główne problemy zgłaszane przez użytkowników w licznych lub długich raportach o błędach.	„Streść wspólne problemy zgłaszane w poniższych raportach z błędami, aby zidentyfikować główne kwestie do rozwiązania".
Artykuły naukowe	Wyodrębnia zwięzłe informacje z długich prac badawczych lub artykułów technicznych, aby użytkownik miał najświeższe informacje na temat najnowszych badań lub technologii.	„Streść główne ustalenia i technologie omawiane w poniższym artykule naukowym".
Rejestr zmian	Umożliwia zrozumienie kluczowych zmian w nowej wersji biblioteki lub narzędzia na podstawie długich rejestrów zmian.	„Streść kluczowe zmiany w poniższym dzienniku zmian wersji 1.1.2".
Komunikacja e-mailowa	Wyodrębnia kluczowe punkty dyskusji lub decyzji z długich wymian informacji poprzez e-mail.	„Streść główne punkty dyskusji z poniższych informacji wymienianych poprzez e-mail".

Innym rodzajem streszczania jest **modelowanie tematów** (ang. *topic modeling*), w którym model statystyczny odkrywa abstrakcyjne „tematy" występujące w zbiorze dokumentów. Oto kilka promptów modelowania tematów dla programistów:

Prompt:

```
Zidentyfikuj główne tematy omawiane w poniższym tekście: {tekst}
```

Prompt:

```
Wyodrębnij słowa kluczowe z poniższego tekstu, aby wywnioskować główne tematy: {tekst}
```

Prompt:

```
Zaproponuj tagi dla poniższego tekstu na podstawie jego treści: {tekst}
```

Klasyfikacja tekstu

Klasyfikacja tekstu polega na dostarczeniu komputerowi zbioru tekstów, które uczy się oznaczać odpowiednimi etykietami. Jednym z jej rodzajów jest **analiza wydźwięku tekstu**: na przykład dla listy postów w mediach społecznościowych model LLM określa, które mają pozytywną, a które negatywną konotację. Dla programistów analiza wydźwięku może być przydatnym narzędziem do oceny opinii użytkowników na temat aplikacji.

Kup książkę Poleć książkę

Oto kilka przykładowych promptów:

Prompt:

```
Czy możesz przeanalizować te recenzje klientów i powiedzieć mi, czy ogólny wydźwięk jest
raczej pozytywny, negatywny czy neutralny? {tekst}
```

Prompt:

```
Oto wątek z naszego forum użytkowników dotyczący najnowszej aktualizacji. Czy mógłbyś
podsumować ogólny wydźwięk? {tekst}
```

Prompt:

```
Zebrałem listę opinii z naszej strony w sklepie z aplikacjami. Czy możesz sklasyfikować
komentarze według wydźwięku? {tekst}
```

Prompt:

```
Oceń wydźwięk tych komentarzy na blogu dotyczących ogłoszenia naszego produktu. Jaka jest
ogólna opinia? {tekst}
```

Zalecenia

Możesz poinstruować model LLM, aby dostarczył rekomendacje dotyczące kodu. Programiści mogą wykorzystać takie informacje zwrotne do poprawy jakości odpowiedzi w takich działaniach jak usuwanie błędów, udoskonalanie kodu czy skuteczniejsze korzystanie z interfejsów API.

Oto kilka przykładowych promptów, których możesz użyć:

Prompt:

```
Poniższy fragment kodu generuje wyjątek NullPointerException, gdy próbuję wywołać
<Metoda()>. Czy możesz pomóc zidentyfikować potencjalną przyczynę i zasugerować rozwiązanie?
```

Prompt:

```
Oto funkcja, którą napisałem do sortowania listy liczb całkowitych. Czy możesz polecić
jakieś optymalizacje, aby ta funkcja działała szybciej lub była bardziej czytelna?
```

Zalecenia modelu LLM mogą być potężnym akceleratorem w Twojej pracy, znacząco oszczędzając czas i dostarczając pomysłów, o których mogłeś nie pomyśleć. Ta technika jest szczególnie przydatna w radzeniu sobie ze skomplikowanymi zadaniami pełnymi niuansów.

Jednakże są pewne wady. Jednym z potencjalnych problemów jest to, że model LLM może zbyt uprościć odpowiedzi i pominąć niuanse. Dodatkowo pamiętaj, że wiedza modelu jest zamrożona na określonym etapie, więc może nie być na bieżąco z najnowszymi informacjami i trendami.

Rekomendacje są sposobem na rozpoczęcie pracy. Ale warto, abyś dogłębnie je zbadał i samodzielnie poszukał więcej informacji, aby uzyskać pełny obraz.

Tłumaczenie

Lokalizacja polega na dostosowaniu oprogramowania do norm językowych i kulturowych szczególnych dla danego regionu. Pozwala to Twojemu oprogramowaniu „mówić” lokalnym

Kup książkę Poleć książkę

językiem i rozumieć regionalną specyfikę, co jest kluczowe dla poszerzenia rynku i zacieśnienia relacji z użytkownikami. Może to przynieść wiele korzyści: użytkownicy są szczęśliwsi, ponieważ oprogramowanie wydaje się skrojone na miarę, a zadowoleni użytkownicy mogą oznaczać lepsze wyniki finansowe dla Twojego biznesu.

Lokalizacja może dać przewagę nad konkurencją, kiedy inne funkcje są niewystarczające lub po prostu nie istnieją. Ponadto przez dostosowanie oprogramowania do lokalnych realiów, w tym zgodność z regionalnymi przepisami, nie tylko czynisz swoje oprogramowanie jednym z wielu możliwych wyborów, ale często jedynym wyborem dla danego rynku.

Z drugiej strony lokalizacja może stanowić wyzwanie. Może być kosztowna i czasochłonna. Wymaga pieczołowitej kontroli jakości, aby utrzymać integralność oprogramowania w różnych językach. Dodatkowo oprogramowanie wciąż jest rozwijane. To ciągły cykl aktualizacji i nowych funkcji, z których każda może wymagać własnych działań lokalizacyjnych. Ten ciągły proces dodaje warstwy złożoności i zwiększa koszty projektu.

Tutaj mogą przyjść z pomocą modele LLM. Zaawansowane systemy są zdolne do tłumaczenia wielu języków. Mogą one pełnić rolę potężnego narzędzia w arsenale programisty. W tabeli 3.2 znajdziesz kilka promptów, których możesz użyć do lokalizacji.

Tabela 3.2. Przykładowe prompty dla tłumaczeń

Rodzaj zadania	Opis	Przykładowy prompt
Tłumaczenie tekstu interfejsu użytkownika	Tłumaczy przyciski, elementy menu, komunikaty o błędach, okna dialogowe itp.	„Przetłumacz następujący tekst interfejsu na język francuski: *Zapisz*, *Wyjdź*, *Plik*, *Edytuj*, *Pomoc*".
Tłumaczenie dokumentacji	Tłumaczy podręczniki użytkownika, pliki pomocy i inną dokumentację.	„Przetłumacz następujący akapit instrukcji obsługi na język hiszpański".
Tłumaczenie komunikatu błędu	Tłumaczy komunikaty o błędach, które mogą być wyświetlane przez oprogramowanie.	„Przetłumacz następujące komunikaty o błędach na język niemiecki: *Plik nieznaleziony*, *Dostęp zabroniony*, *Utracono połączenie z siecią*".
Tłumaczenie wskazówek	Tłumaczy podpowiedzi, które udzielają dodatkowych informacji, gdy użytkownik najedzie kursorem na dany element.	„Przetłumacz następujące wskazówki na język japoński: *Naciśnij, by zapisać*; *Naciśnij, by otworzyć nowy plik*; *Naciśnij, by wydrukować*".

Niemniej jednak ważne jest, aby ostrożnie podchodzić do wielojęzycznych możliwości modeli językowych. Nie są one niezawodne. Modele te mogą czasami przeoczyć subtelności, idiomatyczne wyrażenia i konteksty kulturowe unikatowe dla danego języka. Niuanse językowe są złożone, a ich poprawny przekład to coś więcej niż tylko dosłowne tłumaczenie — chodzi o przekazanie znaczenia w odpowiedni sposób.

Tłumaczenie specyficznych terminów lub nazw może być trudne, zwłaszcza gdy nie ma ich odpowiednika w innym języku. Następnie pojawia się wyzwanie związane z odpowiednim doborem tonu i stylu. Nie chodzi tylko o same słowa, ale także o to, jak się je wymawia, co może się znacznie różnić w zależności od języka czy kultury.

Kup książkę Poleć książkę

Skorzystanie z pomocy specjalisty językowego do sprawdzenia wyników może zaoszczędzić Ci wielu problemów w przyszłości.

Wprowadzanie treści

Kiedy tworzysz prompty, pomocne jest użycie specjalnych ciągów symboli, takich jak ### lub """, służących do wyraźnego oddzielenia Twoich instrukcji od treści lub informacji, nad którymi chcesz, aby model językowy pracował. Te symbole działają jak granice lub znaczniki, wyraźnie wskazując, gdzie kończą się instrukcje, a gdzie zaczyna się treść.

Zastanówmy się nad scenariuszem, w którym programista potrzebuje pomocy w podsumowaniu kluczowych punktów z obszernej dokumentacji dotyczącej nowego właśnie integrowanego interfejsu API. Oto jak można skonstruować prompt:

Prompt:

```
Wyodrębnij kluczowe kroki wdrożenia interfejsu API z poniższego tekstu:
Dokumentacja: """
{Tekst dokumentacji API}
"""
```

Używanie ograniczników """ to sprytny sposób na oddzielenie instrukcji od tekstu dokumentacji API. Daje to modelowi LLM jaśniejszy obraz tego, co należy zrobić, co z kolei zwiększa szanse na uzyskanie klarownego podsumowania głównych kroków. Ponadto te ograniczniki porządkują podpowiedź, czyniąc ją łatwiejszą do odczytania, co jest prawdziwym wybawieniem dla dłuższych lub bardziej złożonych tekstów wejściowych.

Format

W swoim prompcie możesz określić format odpowiedzi zwracanej przez model LLM. Oto przykład:

Prompt:

```
Utwórz funkcję w języku Python, która pobiera listę obiektów użytkownika (każdy obiekt
składa się z ID użytkownika i imienia) i zwraca obiekt w formacie JSON, który dopasowuje
numery ID do imion. Zwróć odpowiedź w formacie JSON.
```

ChatGPT:

```
{
   "code": "def map_user_ids_to_names(users):\n
      user_map = {user['id']: user['name'] for user in users}\n
      return user_map",
   "example_usage": {
      "input": [
         {"id": 1, "name": "Alice"},
         {"id": 2, "name": "Bob"},
         {"id": 3, "name": "Charlie"}
      ],
      "output": {
         "1": "Alice",
         "2": "Bob",
```


Kup książkę Poleć książkę

```
            "3": "Charlie"
        }
    }
}
```

Istnieją inne sposoby formatowania danych wyjściowych. Niektóre z nich przedstawia tabela 3.3.

Tabela 3.3. Prompty do formatowania danych wyjściowych

Rodzaj formatowania	Przykładowy prompt
Tabela	„Utwórz tabelę porównującą składnię, wydajność i przypadki użycia dla języków Python, Java i C++".
Lista	„Wylistuj kroki, jakie należy podjąć w celu rozwiązania problemu wolno ładujących się stron".
Znacznik/HTML	„Wyjaśnij różnice między metodami żądań HTTP GET i POST w języku Markdown".
Hierarchia tekstu	„Podaj ustrukturyzowany zarys cyklu życia oprogramowania (SDLC) wraz z etapami i kluczowymi zadaniami na każdym etapie".
Formatowanie LaTeX	„Podaj złożoność czasową algorytmu przeszukiwania binarnego w formacie LaTeX".

Za pomocą promptu możesz również określić długość odpowiedzi. Możesz poprowadzić model językowy, wydając na przykład instrukcję „Podaj krótkie streszczenie" lub „Napisz szczegółowe wyjaśnienie". Możesz też być bardziej precyzyjny, mówiąc, że odpowiedź nie powinna przekraczać 300 słów. Model językowy może przekroczyć podany przez Ciebie limit słów, ale przynajmniej będzie się trzymał ogólnych ram.

Sprawdzone praktyki

Przyjrzyjmy się teraz kilku najlepszym praktykom dotyczącym tworzenia promptów, które pomogą uzyskać oczekiwane odpowiedzi. Nie traktuj jednak tych sugestii jak niepodważalnej prawdy. Te wskazówki są raczej ogólnymi poradami — które mogą być subiektywne — niż twardymi zasadami. Im więcej czasu spędzisz na rozmowach z dużym modelem językowym, tym większe prawdopodobieństwo, że sam odkryjesz sposoby zadawania pytań, które będą działać. To wszystko jest częścią podróży po inżynierii promptów.

Bądź dokładny

Tworzenie odpowiednich promptów jest jak znalezienie odpowiedniego tonu w rozmowie i może być najważniejszym krokiem do nawiązania udanej interakcji z systemami generującymi tekst. Im więcej szczegółów, tym lepiej. Twoje prompty muszą być jasne i jednoznaczne. W przeciwnym razie model LLM może przyjąć błędne założenia lub nawet halucynować.

Najpierw przyjrzyjmy się kilku promptom, które są zbyt ogólne.

Prompt:

```
Opracuj funkcję zwiększającą bezpieczeństwo danych.
```

Prompt:

```
Czy możesz stworzyć narzędzie do automatyzacji procesu?
```


Kup książkę Poleć książkę

Prompt:

```
Optymalizuj kod.
```

Prompt:

```
Potrzebujemy funkcji do przetwarzania transakcji.
```

Prompty pokazane poniżej są znacznie bardziej szczegółowe i powinny dać lepsze rezultaty:

Prompt:

```
Opracuj funkcję w Pythonie do wyciągania dat z łańcuchów znaków. Funkcja powinna
obsługiwać formaty YYYY-MM-DD, MM/DD/YYYY oraz Month DD, YYYY. Powinna zwracać obiekt
datetime. Przygotuj skrypt, który zademonstruje działanie funkcji, obsługując poprawnie
przynajmniej trzy przykłady każdego z formatów, wraz z dokumentem wyjaśniającym wszelkie
zależności, logikę używaną w funkcji i instrukcje dotyczące uruchomienia skryptu.
```

Prompt:

```
Opracuj zapytanie SQL do pobierania z naszej bazy danych listy klientów, którzy dokonali
zakupów powyżej 500 złotych w ostatnim kwartale 2023 roku. Zapytanie powinno zwracać
pełne imię i nazwisko klienta, jego adres e-mail, całkowitą kwotę wydaną oraz datę
ostatniego zakupu. Wyniki powinny być posortowane według kwoty całkowitej w porządku
malejącym. Proszę upewnić się, że zapytanie jest zoptymalizowane pod kątem wydajności.
```

Skróty i terminy techniczne

Podczas tworzenia promptu ważne jest, aby precyzyjnie dobierać terminy techniczne i skróty. Żargon techniczny często oznacza różne rzeczy w różnych kontekstach i może prowadzić do nieprzydatnych odpowiedzi. Dlatego warto rozwijać skróty i podawać jasne definicje lub wyjaśnienia używanych terminów technicznych.

Załóżmy, że używasz ChatGPT do rozwiązania problemu z połączeniem z bazą danych. Źle sformułowany prompt może brzmieć:

Prompt:

```
Mam problemy z połączeniem się z DB. Jak to naprawić?
```

W tym prompcie skrót *DB* jest niejasny, ponieważ może odnosić się do różnych systemów baz danych, takich jak MySQL czy PostgreSQL, a natura problemu z połączeniem nie jest sprecyzowana. Prompt dający lepsze rezultaty brzmiałby:

Prompt:

```
Mam problem z przekroczeniem limitu czasu połączenia podczas próby połączenia z bazą
danych PostgreSQL za pomocą JDBC. Jak mogę to rozwiązać?
```

Ten prompt jasno określa, jaki system baz danych jest używany, metodę połączenia i napotkany problem.

Mark Twain napisał kiedyś (*https://oreil.ly/ZL9d6*): „Różnica między prawie właściwym słowem a właściwym słowem jest naprawdę duża. To różnica między świetlikiem a błyskawicą". W pewnym sensie to samo można powiedzieć o pisaniu promptów.

Kup książkę Poleć książkę

Uczenie bez przykładów i z kilkoma przykładami

W przypadku **uczenia bez przykładów** podajesz jeden prompt i otrzymujesz oczekiwaną odpowiedź. Często to działa dobrze. Jednak z uwagi na złożoność języków programowania i frameworków zdarzają się sytuacje, gdy musisz delikatnie nakierować model LLM.

Możesz to zrobić za pomocą **uczenia z kilkoma przykładami**. Model LLM potrafi zrozumieć i wykonać zadania na podstawie bardzo niewielkiej liczby przykładów lub danych do treningu. Jest to znaczna przewaga nad tradycyjnymi modelami uczenia maszynowego, które do prawidłowego wykonania zadania mogą wymagać dużej ilości danych treningowych. Zdolność modeli LLM wynika głównie z rozległego wstępnego treningu na różnorodnych tekstach z internetu, przeprowadzanego, zanim model zostanie dostrojony do konkretnego zadania.

Przyjrzyjmy się promptowi z kilkoma przykładami. Rozważmy scenariusz, w którym chcemy wygenerować funkcję normalizującą daną listę liczb. Funkcja będzie skalować wartości w liście do przedziału [0, 1]. W instrukcjach uwzględnimy kilka przykładów wejść i znormalizowanych wyjść.

Prompt:

```
W oparciu o następujące przykłady normalizowania listy liczb do zakresu [0, 1]:
1. Wejście: [2, 4, 6, 8] Wyjście: [0, 0.3333, 0.6667, 1]
2. Wejście: [5, 10, 15] Wyjście: [0, 0.5, 1]
3. Wejście: [1, 3, 2] Wyjście: [0, 1, 0.5]
wygeneruj funkcję w Pythonie, która na wejściu pobiera listę liczb i zwraca listę
znormalizowanych liczb.
```

ChatGPT „nauczy się” z danych i wygeneruje kod.

```
def normalize(numbers):
   min_value = min(numbers)
   max_value = max(numbers)
   if min_value == max_value:
      return [0] * len(numbers)  # Handle the case where all numbers are the same
   else:
      return [(x - min_value) / (max_value - min_value) for x in numbers]
#  Test the function with one of the examples
print(normalize([2, 4, 6, 8])) # Output: [0.0, 0.3333333333333333,
                               # 0.6666666666666666, 1.0]
```

Słowa kluczowe

Koncepcja **słów kluczowych** (ang. *leading words*) odnosi się do specyficznych słów kluczowych lub zwrotów, które mogą nakierować modele LLM na stworzenie określonego rodzaju wyniku. Czasami można osiągnąć pożądany rezultat, używając tylko jednego słowa kluczowego. Oto przykład:

Prompt:

```
# Utwórz prostą funkcję w języku Python, która
# 1. Prosi mnie o podanie temperatury w stopniach Fahrenheita
# 2. Zamienia temperaturę w stopniach Fahrenheita na temperaturę w stopniach Celsjusza.
def
```


Kup książkę Poleć książkę

Użycie słowa *def* jako słowa kluczowego informuje model, że powinien rozpocząć pisanie funkcji w Pythonie. Tabela 3.4 zawiera więcej przykładów słów kluczowych.

Tabela 3.4. Przykłady słów kluczowych

Kontekst	Słowo kluczowe
Funkcja JavaScriptu	`Function`
Element HTML	`<button`
Styl CSS	`P {`
Metoda dodawania wpisu do bazy SQL	`INSERT INTO`
Tworzenie metody w języku Java	`Public`

Łańcuch myśli

W 2022 roku badacze z firmy Google wprowadzili podejście do pisania promptów o nazwie **łańcuch myśli** (CoT — ang. *Chain of Thoughts*) w swojej pracy zatytułowanej *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models* (*https://arxiv.org/abs/2201.11903*). Ta metoda zwiększa zdolności rozumowania modelu LLM poprzez rozbicie złożonego problemu na pojedyncze kroki. W rzeczywistości to podejście przypomina uczenie z kilkoma przykładami, które umożliwia delikatne nakierowanie modelu.

Prompt z łańcuchem myśli może być bardzo przydatny w generowaniu kodu oprogramowania. Przyjrzyjmy się przykładowi. Załóżmy, że chcesz stworzyć aplikację internetową z funkcją rejestracji i logowania użytkowników przy użyciu Flaska, frameworka webowego w Pythonie. Tabela 3.5 pokazuje kroki pisania promptu z łańcuchem myśli.

Tabela 3.5. Przykłady promptów z łańcuchem myśli

Opis działania	Prompt
Zrozumienie wymagań	„Muszę stworzyć aplikację webową z użyciem frameworka Flask. Aplikacja powinna mieć możliwość rejestracji użytkownika i funkcję logowania. Od czego powinienem zacząć?"
Konfiguracja aplikacji Flask	„Zacznijmy od podstawowej aplikacji we frameworku Flask. Jak to mogę zrobić?"
Utworzenie modelu użytkownika	„Teraz, gdy już mam podstawową aplikację we frameworku Flask, muszę utworzyć model użytkownika do obsługi rejestracji i logowania. Jaką strukturę powinien mieć ten model?"
Implementacja rejestracji	„Gdy już mam model, jak mogę zaimplementować stronę do rejestracji z wszystkimi niezbędnymi polami?"
Implementacja logowania	„Teraz przejdźmy do tworzenia strony logowania. Jak mogę zapewnić bezpieczne logowanie?"
Zarządzanie sesją	„Gdy użytkownik się już zaloguje, jak powinienem zarządzać sesją użytkownika, by użytkownik pozostał cały czas zalogowany podczas poruszania się po aplikacji?"
Implementacja wylogowania	„Na koniec, jak mogę zaimplementować funkcję wylogowania, by użytkownicy mogli bezpiecznie wyjść z aplikacji?"

Kup książkę Poleć książkę

Pytania kluczowe

Pytania kluczowe (ang. *leading questions*) w promptach często mogą prowadzić do błędnych odpowiedzi podawanych przez model LLM. Lepiej pozostać neutralnym i bezstronnym. Co więcej, warto nic nie zakładać z góry. Lepiej wszystko dokładnie wyjaśnić.

Przykład pytania kluczowego:

> Prompt:
>
> ```
> Czy to prawda, że migracja do architektury mikrousług zawsze poprawia skalowalność
> systemu?
> ```

Bardziej zrównoważony prompt to:

> Prompt:
>
> ```
> Jakie są zalety i potencjalne wyzwania migracji do architektury mikrousług w kontekście
> skalowalności systemu?
> ```

Prośba o przykłady i analogie

Załóżmy, że nie znasz pojęcie dziedziczenia w programowaniu obiektowym. Do ChatGPT wpisujesz taki prompt:

> Prompt:
>
> ```
> Wyjaśnij dziedziczenie stosowane w programowaniu obiektowym.
> ```

Otrzymasz szczegółową odpowiedź. Ale może chcesz uzyskać coś, co będzie łatwiejsze do zrozumienia. Dobrym sposobem jest poproszenie modelu LLM o analogię:

> Prompt:
>
> ```
> Wyjaśnij dziedziczenie stosowane w programowaniu obiektowym przy użyciu analogii.
> ```
>
> ChatGPT:
>
> ```
> Pomyśl o dziedziczeniu jak o drzewie genealogicznym, gdzie dzieci dziedziczą pewne cechy
> i właściwości od swoich rodziców, a potencjalnie także od dziadków.
> ```

Następnie ChatGPT podaje więcej szczegółów, dalej bazując na tej analogii, aby wyjaśnić kluczowe elementy dziedziczenia.

Zmniejszanie efektu halucynacji

W rozdziale 2. dowiedzieliśmy się, że wysyłanie promptów do modelu LLM może prowadzić do odpowiedzi, które są **halucynacjami**, co oznacza, że generowana zawartość jest fałszywa lub wprowadza w błąd, przy czym model językowy wyraża odpowiedź tak, jakby była prawdziwa. Halucynacje mogą być szczególnym wyzwaniem dla rozwoju oprogramowania, gdzie wymagana jest dokładność.

W tym rozdziale dowiesz się, jak można złagodzić ten problem, ale nawet dobrze skonstruowany prompt może wywołać halucynacje. Istnieje wiele powodów tego zjawiska:

Kup książkę Poleć książkę

Brak weryfikacji z rzeczywistością

Modele LLM generują odpowiedzi na podstawie wyuczonych wzorców z danych treningowych bez możliwości weryfikacji dokładności lub rzeczywistych informacji.

Nadmierne dopasowanie i zapamiętywanie

Modele LLM mogą zapamiętać nieprawidłowe lub wprowadzające w błąd informacje zawarte w swoich zbiorach treningowych, zwłaszcza jeśli takie dane są powtarzalne lub powszechne.

Stronniczość danych treningowych

Jeśli dane treningowe zawierają uprzedzenia, nieścisłości lub fałszywe informacje, model prawdopodobnie odtworzy je w swoich wynikach.

Prognozowanie i spekulacja

Czasami modele LLM mogą prognozować ze wzorców, które widziały w danych, aby generować informacje na tematy lub pytania, które nie były wystarczająco omówione w danych treningowych.

Brak kontekstu lub niewłaściwa interpretacja

Modele LLM mogą błędnie interpretować kontekst albo nie mają go w ogóle, dlatego nie są w stanie dokładnie odpowiedzieć na niektóre prompty. Modele językowe mogą nie w pełni rozumieć niuanse lub implikacje niektórych zapytań.

Slang i idiomy

Slang i idiomy mogą powodować niejednoznaczności, które z kolei mogą prowadzić do niewłaściwej interpretacji zamierzonego znaczenia, zwłaszcza jeśli model podczas treningu nie widział wystarczającej liczby przykładów tego slangu lub idiomu w kontekście.

Jak zatem zmniejszyć zjawisko halucynacji? Ważne jest zwłaszcza, aby unikać zadawania pytań otwartych takich jak to:

Prompt:

```
Jakie są różne sposoby optymalizacji bazy danych?
```

Tego typu podpowiedź zachęca model LLM do spekulacji lub nadmiernego uogólniania. Model językowy może również błędnie interpretować intencje pytania lub oczekiwany format odpowiedzi, co prowadzi do odpowiedzi, które odbiegają od tematu lub zawierają wymyślone informacje. Może nawet dojść do kaskady halucynacji.

Jedną ze skutecznych technik jest podanie z góry zestawu określonych opcji i poproszenie sztucznej inteligencji o wybór jednej z nich. Na przykład wcześniejszy prompt można przeformułować w następujący sposób:

Prompt:

```
Która z poniższych metod służy do optymalizacji bazy danych: indeksowanie,
defragmentacja czy kompresja?
```

Innym przykładem może być poproszenie modelu LLM o określony rodzaj odpowiedzi. Oto przykładowy prompt:

Prompt:

```
Czy poniższa składnia jest poprawna dla inicjalizacji tablicy w języku Java? Odpowiedz
„tak" lub „nie".
```


Kup książkę Poleć książkę

W prompcie możesz również zawrzeć wiele kroków, aby lepiej poprowadzić model przez ustrukturyzowany proces i zawęzić możliwości zboczenia z kursu.

Prompt:

```
Krok 1: Stwórz generator ciągu Fibonacciego.
Krok 2: Użyj metody iteracyjnej.
Krok 3: Napisz funkcję w Pythonie o nazwie generate_fibonacci, która przyjmuje liczbę
całkowitą n jako argument.
Krok 4: Funkcja zwraca pierwsze n liczb w ciągu Fibonacciego jako listę.
```

Bezpieczeństwo i prywatność

Zachowanie ostrożności w kwestiach bezpieczeństwa i prywatności podczas tworzenia promptu jest kluczowe. W rzeczywistości obowiązek podjęcia odpowiednich środków ostrożności powinien być zapisany w regulaminie firmy. Należy unikać w promptach wszelkich wrażliwych lub osobistych danych, takich jak dane umożliwiające identyfikację osoby (PII). Oto przykład promptu zawierającego takie informacje:

Prompt:

```
Jak naprawić problem z logowaniem zgłoszony przez Johna Doe na adres
john.doe@example.com?
```

Rozsądniej jest użyć czegoś w stylu:

Prompt:

```
Jak rozwiązałbyś problem z logowaniem zgłoszony przez użytkownika?
```

To pozwala zachować prywatne dane w tajemnicy.

Co więcej, w promptach warto unikać ujawniania jakichkolwiek wrażliwych szczegółów systemu. Na przykład:

Prompt:

```
Jak naprawić błąd połączenia z bazą danych na naszym serwerze produkcyjnym o adresie IP
192.168.1.1?
```

Zamiast tego bezpieczniej jest użyć bardziej ogólnego pytania:

Prompt:

```
Jak naprawić ogólny błąd połączenia z bazą danych?
```

Co więcej, upewnij się, że Twoje prompty przypadkowo nie sugerują podejrzanych praktyk. Taki prompt jest w porządku z punktu widzenia bezpieczeństwa:

Prompt:

```
Jak wykrywać ataki typu SQL injection i im zapobiegać?
```

Ale ten prompt już może sugerować złe intencje:

Prompt:

```
Jak wykorzystywać luki SQL na stronie internetowej?
```


Kup książkę Poleć książkę

Poza przestrzeganiem zasad bezpieczeństwa i prywatności ważne jest również uwzględnianie różnorodności i inkluzywności przy tworzeniu promptu. Kluczowe jest zrozumienie uprzedzeń, które często odzwierciedlają dane treningowe. Dobrym pomysłem jest używanie neutralnego i włączającego języka, aby unikać jakichkolwiek dyskryminujących lub wykluczających wyrażeń w podpowiedziach. Dodatkowo może być pomocne uzyskanie opinii od różnorodnej grupy osób na temat tworzenia promptów. To nie tylko zwiększa uczciwość i inkluzywność podczas interakcji z modelem LLM, ale także pomaga dokładniej i bardziej wszechstronnie zrozumieć omawiane tematy.

Autonomiczne agenty sztucznej inteligencji

Widzieliśmy, jak można nakłonić modele LLM do wyznaczania kroków procesu. To jest istota generowania kodu.

Ale agenty sztucznej inteligencji mogą wynieść to na wyższy poziom. Nie tylko wykonują polecenia, ale także kreatywnie współpracują z modelem LLM, aby opracować plan działania dla każdego celu, który im wyznaczysz. Ponadto korzystają ze specjalistycznych baz danych takich jak Pinecone i Chroma DB. Radzą sobie ze złożonymi osadzeniami słów rozumianymi przez modele.

Autonomiczne agenty sztucznej inteligencji opierają się na badaniach akademickich i zazwyczaj są częścią projektów open source. Ich prawdziwa moc tkwi w automatyzacji. Zobaczmy na przykładzie, jak to działa. Załóżmy, że postawisz cel w następujący sposób:

Prompt:

```
Stwórz podstawową aplikację pogodową z systemem logowania użytkowników.
```

Tabela 3.6 pokazuje proces, przez który może przejść autonomiczny agent.

Tabela 3.6. Proces dla autonomicznego agenta

Faza	Zadania
Projektowanie	Zaprojektuj interfejs użytkownika.
	Naszkicuj podstawowy układ interfejsu użytkownika.
	Wybierz motyw kolorystyczny i czcionki.
	Zaprojektuj ikony i inne elementy graficzne.
Integracja API dla danych pogodowych	Wyszukaj w internecie solidne API dla danych pogodowych.
	Określ punkty danych, które mają być wyświetlane.
	Napisz kod pobierający i aktualizujący dane pogodowe.
Funkcja wyboru lokalizacji	Utwórz pasek wyszukiwania lub listę rozwijalną dla użytkowników, by mogli wybrać swoją lokalizację.
	Połącz to z kodem API.
Obsługa błędów	Obsługuj błędy takie jak błąd API lub niepoprawna podana lokalizacja.
Priorytetyzowanie zadań	Najpierw skonfiguruj integrację API.
	Skup się na interfejsie użytkownika.
	Pracuj nad funkcjonalnością wyboru lokalizacji i obsługą błędów.

Kup książkę Poleć książkę

Tabela 3.6. Proces dla autonomicznego agenta (ciąg dalszy)

Faza	Zadania
Iteracja	Przejrzyj wygenerowany kod oraz bieżący stan pulpitu nawigacyjnego z pogodą.
	Określ pozostałe zadania lub nowe zadania, które pojawiły się podczas realizacji.
	Powtórz kroki tworzenia i priorytetyzacji.

Ta technologia jest teraz wiodąca i wiąże się z nią wiele nadziei. Jednakże nie jest pozbawiona słabości.

Pożeracze zasobów

Agenty mogą zużywać duże ilości mocy obliczeniowej. Może to obciążać procesory i bazy danych, prowadząc do dłuższych czasów oczekiwania, mniejszej niezawodności i spadku wydajności z czasem.

Zawieszanie się w nieskończonych pętlach

Czasami agenty kręcą się w kółko z powodu braku postępu lub powtarzającego się systemu nagród.

Etap eksperymentalny

Agenty mogą być niedopracowane. Mogą zawierać kilka błędów lub wykazywać niespodziewane zachowania i nie być jeszcze gotowe do zaawansowanych zastosowań.

Amnezja

Agenty mogą po prostu zapominać pewnych kroków lub instrukcji.

Trudności w obsłudze dużej liczby zadań

Masz długą listę zadań? To może być problematyczne dla agentów.

Rozpraszanie się zbędnymi szczegółami

Agenty mogą się rozpraszać mało istotnymi rzeczami, co może utrudnić im wybór narzędzi do użycia.

Kolejną innowacją wspierającą modele LLM jest **pobieranie rozszerzonej generacji** (RAG — ang. *retrieval augmented generation*). Dzięki RAG aplikacja oparta na generatywnej sztucznej inteligencji — na przykład napisana we frameworku LangChain — uzyskuje dostęp do zewnętrznych źródeł danych, zazwyczaj wektorowych baz danych. Takie zewnętrzne źródła danych dostarczają konkretniejszej wiedzy, co powinno poprawić odpowiedzi modelu LLM.

Pobieranie rozszerzonej generacji może być szczególnie użyteczne przy obsłudze złożonych zadań związanych z rozwojem oprogramowania, na przykład w takich scenariuszach:

Naprawa błędów i usterek

Gdy programiści napotykają błędy lub usterki, RAG wyszukuje poprawki i obejścia z internetu, przeszukując miejsca takie jak fora czy bazy danych błędów. Pobieranie rozszerzonej generacji może pomóc w przygotowaniu dedykowanych rozwiązań lub poprawek kodu dostosowanych do Twojego problemu.

Kup książkę Poleć książkę

Urozmaicanie przeglądów kodu

RAG może czerpać najlepsze praktyki kodowania, standardy i obowiązkowe zasady z wewnętrznych zasobów firmy. Oznacza to, że może pomóc usprawnić przeglądy kodu, dostarczając wskazówki i triki, które sprawią, że kod będzie lepszy.

Szybsze testowanie

Gdy nadchodzi czas na sprawdzenie Twojego kodu, RAG może być Twoim wsparciem. Dzięki pobieraniu rozszerzonej generacji możesz mieć różnorodne scenariusze i wzorce testowe dostosowane do potrzeb Twojego projektu i szybko wdrażać przypadki lub skrypty testowe.

W niektórych przypadkach model LLM wskaże, że istnieje problem z Twoim promptem. Przypuśćmy, że napiszesz:

Prompt:

```
Napisz REST API, używając języka asemblera.
```

Oto jak odpowiedział ChatGPT:

ChatGPT:

```
Tworzenie REST API przy użyciu języka asemblera jest wysoce niekonwencjonalne
i złożone ze względu na niskopoziomowy charakter języków asemblera. Te języki
są zazwyczaj używane do zadań wymagających precyzyjnej kontroli nad
sprzętem lub w scenariuszach, gdzie wydajność jest kluczowym problemem.
```

Podsumowanie

Tworzenie idealnego promptu to mieszanka nauki z odrobiną kreatywności. Chodzi o znalezienie odpowiednich składników — trochę kreatywności, odrobiny intuicji i strukturalnego podejścia — aby wyczarować prompty, które sprawią, że modele LLM dostarczą to, czego potrzebujesz. Nie ma magicznego przepisu, ale jeśli będziesz wyrażał się jasno, dodasz kilka przykładów i dobrze sformułujesz swoje prompty, to będziesz na dobrej drodze do uzyskania lepszych odpowiedzi.

To jest proces. Próbujesz czegoś, obserwujesz, jak to działa, wprowadzasz poprawki i próbujesz ponownie. I jak w przypadku każdej umiejętności, im więcej pracujesz z różnymi tematami i zadaniami, tym lepszy się stajesz.

Kup książkę Poleć książkę


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

Ta książka zaoszczędzi Ci wielu godzin prób i błędów!
Jonathan Ellis, współzałożyciel firmy DataStax

To już się dzieje! Narzędzia oparte na sztucznej inteligencji wykonują monotonne zadania i zajmują się złożonymi szczegółami kodu. W tym czasie programista może się skupić na rozwiązywaniu problemów i innowacjach. AI w takim tandemie ogrywa rolę zaufanego pomocnika, wyręczającego człowieka w zawiłych lub nużących aspektach kodowania. Efekt? Imponujący wzrost produktywności!

Ta praktyczna książka ułatwi Ci optymalne używanie narzędzi AI na wszystkich etapach tworzenia oprogramowania. Niezależnie od Twojego doświadczenia nauczysz się korzystać z szerokiej gamy rozwiązań: od dużych modeli językowych ogólnego przeznaczenia (ChatGPT, Gemini i Claude) po systemy przeznaczone do kodowania (GitHub Copilot, Tabnine, Cursor i Amazon CodeWhisperer). Poznasz również metodykę programowania modułowego, która efektywnie współgra z technikami pisania promptów do generowania kodu. W książce znajdziesz także najlepsze sposoby zastosowania uniwersalnych modeli LLM w nauce języka programowania, wyjaśnianiu kodu lub przekładaniu go na inny język programowania.

Najważniejsze zagadnienia:

- możliwości narzędzi opartych na AI, przeznaczonych do tworzenia kodu
- zalety i wady popularnych systemów
- korzystanie z ogólnych modeli językowych podczas kodowania
- narzędzia oparte na AI w cyklu życia oprogramowania
- inżynieria promptów podczas tworzenia oprogramowania
- realizacja żmudnych zadań, takich jak pisanie wyrażeń regularnych

Tom Taulli jest konsultantem, autorem książek i trenerem. Prowadził kursy informatyczne dla wydawnictwa O'Reilly, Uniwersytetu Kalifornijskiego w Los Angeles (UCLA) i platformy edukacyjnej Pluralsight; uczestników zajęć uczył używać Pythona do tworzenia modeli głębokiego uczenia i uczenia maszynowego. Prowadził również kursy na temat przetwarzania języka naturalnego.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-2082-8
9 788328 920828
Cena: 79,00 zł