Valentina Alto

LLM w projektowaniu oprogramowania

Tworzenie inteligentnych aplikacji i agentów z wykorzystaniem dużych modeli językowych

‹packt›

Helion

Tytuł oryginału: Building LLM Powered Applications: Create intelligent apps and agents with large language models

Tłumaczenie: Robert Górczyński

ISBN: 978-83-289-1894-8

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

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

Printed in Poland.

Spis treści

O autorze 9

O korektorach merytorycznych 10

Wprowadzenie 11

ROZDZIAŁ 1.
Wprowadzenie do dużych modeli językowych 19

Czym jest duży model podstawowy i duży model językowy? 20
Przesunięcie paradygmatu sztucznej inteligencji — wprowadzenie do modeli podstawowych 21
Budowa dużego modelu językowego 23
Najpopularniejsze architektury oparte na transformerach dużego modelu językowego 30
Wczesne eksperymenty 30
Wprowadzenie do architektury transformera 31
Trenowanie dużego modelu językowego i jego ocena 37
Trenowanie dużego modelu językowego 37
Ocena modelu 40
Modele podstawowe kontra modele dostosowane do własnych potrzeb 43
Jak można dostosować model do własnych potrzeb? 43
Podsumowanie 46
Odwołania 46

ROZDZIAŁ 2.
Duże modele językowe w aplikacjach wspomaganych przez sztuczną inteligencję 47

Sposoby, w jakie duże modele językowe zmieniły branżę tworzenia oprogramowania 48
System copilot 49
Wprowadzenie do frameworków koordynowania sztucznej inteligencji umożliwiających osadzanie dużych modeli językowych w aplikacjach 54

Najważniejsze komponenty frameworków koordynowania sztucznej inteligencji ... 54
LangChain ... 57
Haystack ... 59
Semantic Kernel ... 60
Jak wybrać framework? ... 63
Podsumowanie ... 65
Odwołania ... 65

ROZDZIAŁ 3.
Wybór dużego modelu językowego dla aplikacji ... 66

Ogólne omówienie najbardziej obiecujących dużych modeli językowych dostępnych na rynku ... 66
Modele własnościowe ... 67
Modele otwartoźródłowe ... 77
Nie tylko modele językowe ... 82
Kryteria stosowane podczas wyboru właściwego dużego modelu językowego ... 85
Rozważania ... 88
Studium przypadku ... 90
Podsumowanie ... 91
Odwołania ... 92

ROZDZIAŁ 4.
Prompt engineering ... 93

Wymagania techniczne ... 93
Czym jest prompt engineering? ... 94
Reguły prompt engineering ... 94
Jasne instrukcje ... 95
Podziel skomplikowane zadania na mniejsze ... 97
Zapytaj o uzasadnienie ... 100
Wygenerowanie wielu danych wyjściowych, a następnie użycie modelu do wybrania tych najlepszych ... 102
Powtórzenie instrukcji na końcu ... 104
Używanie ograniczników ... 106
Techniki zaawansowane ... 109
Technika few-shots ... 109
Technika chain of thought ... 112
ReAct ... 114
Podsumowanie ... 118
Odwołania ... 118

ROZDZIAŁ 5.
Osadzanie dużych modeli językowych w aplikacjach 120
Wymagania techniczne 120
Wprowadzenie do frameworka LangChain 121
Rozpoczęcie pracy z frameworkiem LangChain 123
Modele i prompty 123
Połączenia danych 126
Pamięć 132
Łańcuch 135
Agenty 140
Praca z dużymi modelami językowymi poprzez platformę Hugging Face Hub 143
Tworzenie tokena dostępu użytkownika Hugging Face Hub 143
Przechowywanie kluczy tajnych użytkownika w pliku .env 146
Rozpoczęcie pracy z otwartoźródłowymi dużymi modelami językowymi 147
Podsumowanie 148
Odwołania 149

ROZDZIAŁ 6.
Tworzenie aplikacji konwersacyjnych 150
Wymagania techniczne 150
Rozpoczęcie pracy z aplikacjami konwersacyjnymi 151
Tworzenie zwykłego bota 152
Dodawanie pamięci 153
Dodawanie wiedzy nieparametrycznej 157
Dodawanie narzędzi zewnętrznych 160
Opracowanie frontendu za pomocą Streamlit 163
Podsumowanie 167
Odwołania 168

ROZDZIAŁ 7.
Używanie dużych modeli językowych do tworzenia silników wyszukiwania i rekomendacji 169
Wymagania techniczne 169
Wprowadzenie do systemu rekomendacji 170
Istniejące systemy rekomendacji 171
Algorytm K najbliższych sąsiadów 172
Rozkład macierzy 173
Sieci neuronowe 176

Jak duże modele językowe zmieniają systemy rekomendacji? 179
Implementowanie systemu rekomendacji opartego na dużym modelu językowym 180
Wstępne przygotowanie danych 181
Tworzenie chatbota rekomendacji w scenariuszu typu zimny start 184
Tworzenie systemu opartego na treści 191
Opracowanie frontendu za pomocą biblioteki Streamlit 196
Podsumowanie 198
Odwołania 199

ROZDZIAŁ 8.
Używanie dużych modeli językowych w połączeniu z danymi strukturyzowanymi 200

Wymagania techniczne 200
Czym są dane strukturyzowane? 201
Rozpoczęcie pracy z relacyjnymi bazami danych 203
Wprowadzenie do relacyjnych baz danych 203
Ogólne omówienie bazy danych Chinook 205
Jak za pomocą Pythona pracować z relacyjną bazą danych? 207
Implementacja aplikacji DBCopilot z użyciem frameworka LangChain 211
Agenty frameworka LangChain i agent SQL 211
Prompt engineering 215
Dodawanie kolejnych narzędzi 218
Opracowanie frontendu za pomocą biblioteki Streamlit 222
Podsumowanie 225
Odwołania 226

ROZDZIAŁ 9.
Praca z kodem źródłowym 227

Wymagania techniczne 227
Wybór odpowiedniego dużego modelu językowego przeznaczonego do pracy z kodem źródłowym 228
Analizowanie i generowanie kodu źródłowego 230
Falcon LLM 231
CodeLlama 234
StarCoder 238
Działanie w charakterze algorytmu 243
Wykorzystanie API Code Interpreter 250
Podsumowanie 256
Odwołania 256

ROZDZIAŁ 10.
Tworzenie wielomodalnych aplikacji wykorzystujących duże modele językowe 258
Wymagania techniczne 258
Dlaczego wielomodalność? 259
Tworzenie agenta wielomodalnego za pomocą frameworka LangChain 260
Opcja 1. Użycie standardowego zbioru narzędzi dla Azure AI Services 262
Rozpoczęcie pracy z AzureCognitiveServicesToolkit 262
Opcja 2. Połączenie narzędzi w postać pojedynczego agenta 276
Narzędzia YouTube i Whisper 277
DALL-E i generowanie tekstu 279
Połączenie wszystkiego w całość 281
Opcja 3. Podejście na stałe zdefiniowane w kodzie z użyciem łańcucha sekwencyjnego 285
Porównanie wszystkich trzech opcji 288
Opracowanie frontendu za pomocą biblioteki Streamlit 290
Podsumowanie 292
Odwołania 293

ROZDZIAŁ 11.
Dostrajanie dużych modeli językowych 294
Wymagania techniczne 295
Czym jest dostrajanie? 295
Kiedy dostrajanie modelu jest konieczne? 298
Rozpoczęcie dostrajania 300
Pobranie zbioru danych 300
Tokenizacja danych 302
Dostrajanie modelu 304
Używanie wskaźników oceny modelu 306
Trenowanie i zapisywanie 310
Podsumowanie 313
Odwołania 313

ROZDZIAŁ 12.
Odpowiedzialna sztuczna inteligencja 314

Czym jest odpowiedzialna sztuczna inteligencja i dlaczego jej potrzebujemy? 314
Architektura odpowiedzialnej sztucznej inteligencji 316
Warstwa modelu 317
Warstwa metapromptu 320
Warstwa interfejsu użytkownika 321
Regulacje prawne dotyczące odpowiedzialnej sztucznej inteligencji 325
Podsumowanie 327
Odwołania 327

ROZDZIAŁ 13.
Najnowsze trendy i innowacje 329

Najnowsze trendy w modelach językowych i generatywnej sztucznej inteligencji 329
GPT-4V(ision) 330
DALL-E 3 332
AutoGen 332
Małe modele językowe 334
Firmy wykorzystujące generatywną sztuczną inteligencję 335
Coca-Cola 335
Notion 336
Malbek 337
Microsoft 337
Podsumowanie 338
Odwołania 340

Rozdział 2

Duże modele językowe w aplikacjach wspomaganych przez sztuczną inteligencję

W poprzednim rozdziale przedstawiłam wprowadzenie do **dużego modelu językowego** (ang. *large language model*, LLM) jako potężnego modelu podstawowego zapewniającego możliwości nie tylko generatywne, ale także w zakresie wnioskowania opartego na zdrowym rozsądku. Być może zastanawiasz się, do czego można wykorzystać takie modele.

W rozdziale zamierzam pokazać, jak te modele zrewolucjonizowały branżę tworzenia oprogramowania, prowadząc do nowej ery aplikacji wspomaganych przez sztuczną inteligencję. Dzięki lekturze rozdziału otrzymasz pełny obraz tego, jak duże modele językowe mogą być osadzane w różnych aplikacjach dzięki wykorzystaniu nowych frameworków koordynowania sztucznej inteligencji, które pojawiają się w branży tworzenia aplikacji wspomaganych przez sztuczną inteligencję.

W rozdziale poruszam następujące zagadnienia:

- sposoby, w jakie duże modele językowe zmieniły branżę tworzenia oprogramowania,
- system copilot,
- wprowadzenie do frameworków koordynowania sztucznej inteligencji umożliwiających osadzanie dużych modeli językowych w aplikacjach.

Sposoby, w jakie duże modele językowe zmieniły branżę tworzenia oprogramowania

Duże modele językowe potwierdziły, że mają wyjątkowo potężne możliwości, od zadań związanych z rozumieniem języka naturalnego (tworzenie podsumowań, rozpoznawanie nazwanych encji i ich klasyfikowanie) po generowanie tekstu, a także od wnioskowania opartego na zdrowym rozsądku po umiejętności związane z burzą mózgów. Jednak same w sobie nie są już tak niewiarygodnie wspaniałe. Jak wyjaśniłam w poprzednim rozdziale, duże modele językowe — i ogólnie rzecz biorąc, **duże modele podstawowe** — stoją za rewolucją w branży tworzenia oprogramowania, jako platformy przeznaczone do opracowywania potężnych aplikacji.

Istotnie, zamiast rozpoczynać pracę zupełnie od początku, obecnie programiści mogą wykonywać wywołania API do dużych modeli językowych dostępnych na serwerach, a także mają możliwość dostosowania tych modeli do własnych potrzeb, jak to już dokładniej omówiłam w poprzednim rozdziale. Ta zmiana pozwala zespołom znacznie łatwiej i efektywniej wykorzystać w budowanych aplikacjach funkcjonalność dostarczaną przez sztuczną inteligencję. To odbywa się na zasadzie podobnej do wcześniejszej zmiany, jaką było przejście od systemów jednozadaniowych do systemów współdzielonych.

Jednak co dokładnie oznacza wykorzystanie dużego modelu językowego w aplikacji? Mamy dwa obszerne aspekty, które należy rozważyć podczas osadzania dużych modeli językowych w budowanych aplikacjach.

- **Aspekt techniczny**. Wskazuje, *jak* przygotować rozwiązanie. Zintegrowanie dużego modelu językowego z aplikacją obejmuje jego osadzenie z wykorzystaniem wywołań API REST oraz zarządzanie nimi za pomocą frameworka koordynowania sztucznej inteligencji. To oznacza przygotowanie komponentów architekturalnych umożliwiających bezproblemową komunikację z dużym modelem językowym za pomocą wywołań API. Ponadto używanie frameworka koordynowania sztucznej inteligencji pomaga w efektywnym zarządzaniu funkcjonalnością dużego modelu językowego w aplikacji oraz jej koordynowaniu, co znacznie dokładniej wyjaśniam w dalszej części rozdziału.
- **Aspekt koncepcyjny**. Wskazuje, *co* należy zrobić. Duże modele językowe zapewniają mnóstwo nowych możliwości, które można wykorzystać w budowanych aplikacjach. Dokładne omówienie tych możliwości znajdziesz w dalszej części książki. Jednym ze sposobów na zapoznanie się z wpływem dużych modeli językowych jest uznanie ich za nową kategorię oprogramowania, którą często określa się mianem **systemu copilot**. Tego rodzaju kategoryzacja podkreśla znaczenie pomocy dużych modeli językowych w usprawnianiu funkcjonalności aplikacji.

W aspekt techniczny zagłębimy się nieco dalej w rozdziale, natomiast w następnym podrozdziale omówię zupełnie nową kategorię oprogramowania, system copilot.

System copilot

System copilot to nowa kategoria oprogramowania, będąca świetnym wsparciem dla użytkowników, który próbują wykonywać skomplikowane zadania. Ta koncepcja, wprowadzona przez firmę Microsoft, znalazła zastosowanie w opracowywanych przez nią produktach, takich jak M365 Copilot i nowa wersja wyszukiwarki internetowej Bing, które obecnie są wspomagane przez GPT-4. Framework wykorzystywany przez te produkty jest obecnie dostępny również dla programistów, którzy mogą go osadzać we własnych aplikacjach.

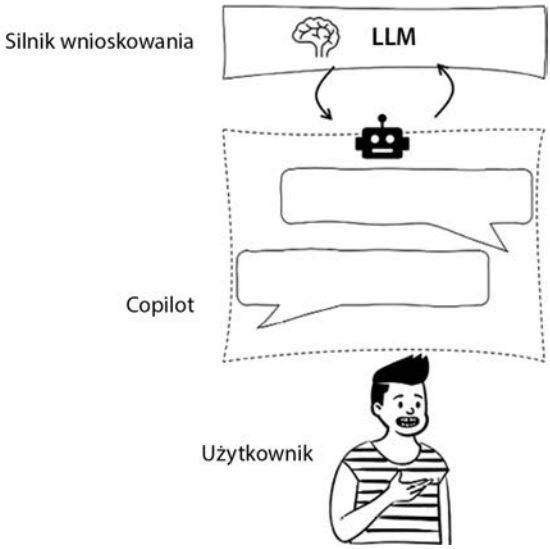
Być może zastanawiasz się, czym dokładnie jest system copilot?

Zgodnie z nazwą copilot to jeden z asystentów sztucznej inteligencji, współpracujących z użytkownikami i pomagających im w wykonywaniu różnych zadań, od pobierania informacji niezbędnych do przygotowania i opublikowania posta na blogu poprzez przeprowadzanie burzy mózgów aż po przegląd kodu i jego generowanie.

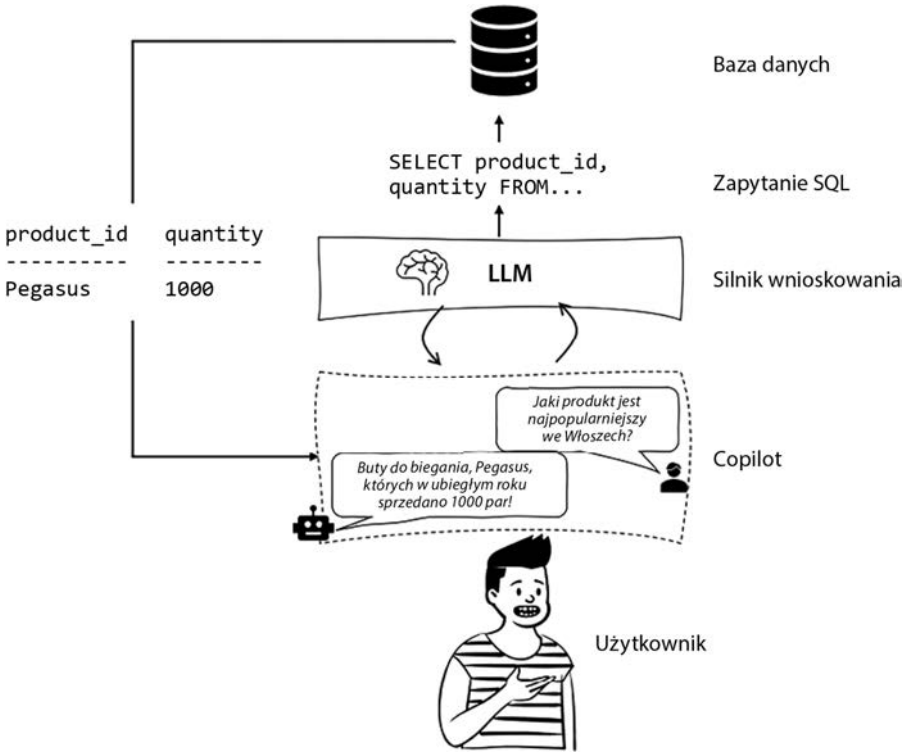
Zapoznaj się teraz z wybranymi i unikatowymi funkcjonalnościami systemu copilota.

- **Podstawą działania copilota są duże modele językowe**, a ogólniej: duże modele podstawowe, co oznacza, że są to silniki wnioskowania, dzięki którym system copilot może działać w sposób „inteligentny". Silnik wnioskowania jest jednym z komponentów copilota, ale bynajmniej nie jedynym. Funkcjonowanie systemu opiera się również na innych technologiach — takich jak aplikacje, źródła danych i interfejsy użytkownika — w celu zagwarantowania użytkownikom przydatnych i ciekawych wrażeń. W sposób graficzny działanie systemu copilota pokazałam na rysunku 2.1.
- **Copilot zaprojektowano w taki sposób, aby udostępniał konwersacyjny interfejs użytkownika**, który pozwala użytkownikom pracować z tym systemem z wykorzystaniem języka naturalnego. Dzięki temu ogranicza się bądź nawet eliminuje luki w wiedzy między skomplikowanymi systemami wymagającymi taksonomii ściśle związanej z daną dziedziną (np. wykonywanie zapytań do danych tabelarycznych wymaga znajomości języków programowania takich jak T-SQL) a użytkownikami. Przykład takiej konwersacji pokazałam na rysunku 2.2.
- **Copilot ma zasięg.** Zatem jest **uziemniony** (ang. *grounding*) i może korzystać jedynie z danych dotyczących konkretnej dziedziny, a tym samym może pomagać tylko w ramach określonej aplikacji lub dziedziny.

  Do przeprowadzenia groundingu używa się frameworka architekturalnego określanego mianem RAG (ang. *retrieval-augmented generation*). Ogólnie rzecz biorąc, to technika poprawiająca dane wyjściowe dużego modelu językowego



Rysunek 2.1. Podstawą działania systemu copilota są duże modele językowe



Rysunek 2.2. Przykład konwersacyjnego interfejsu użytkownika, który pomaga zmniejszyć lukę w wiedzy między użytkownikiem a bazą danych

Definicja

Grounding to proces używania dużych modeli językowych razem z informacjami ściśle związanymi z danym przypadkiem użycia, które są istotne i które nie zostały zdobyte w trakcie trenowania modelu. Znaczenie krytyczne ma zagwarantowanie jakości, dokładności i trafności wygenerowanych danych wyjściowych. Na przykład przyjmujemy założenie, że chcesz otrzymać aplikację wspomaganą przez sztuczną inteligencję, która będzie pomagała w pracy nad bieżącymi artykułami (nieuwzględnionymi w zbiorze danych używanych do trenowania dużego modelu językowego). Ponadto chcesz, aby aplikacja udzieliła odpowiedzi jedynie wtedy, gdy znajduje się ona w tych artykułach. W tym celu duży model językowy trzeba będzie ograniczyć tylko do podanego zbioru artykułów, aby aplikacja udzielała odpowiedzi jedynie w określonych ramach.

dzięki wykorzystaniu podczas generowania odpowiedzi informacji pochodzących z zewnętrznej, sprawdzonej bazy wiedzy. Ten proces pomaga zagwarantować, że wygenerowana treść jest trafna, poprawna i aktualna.

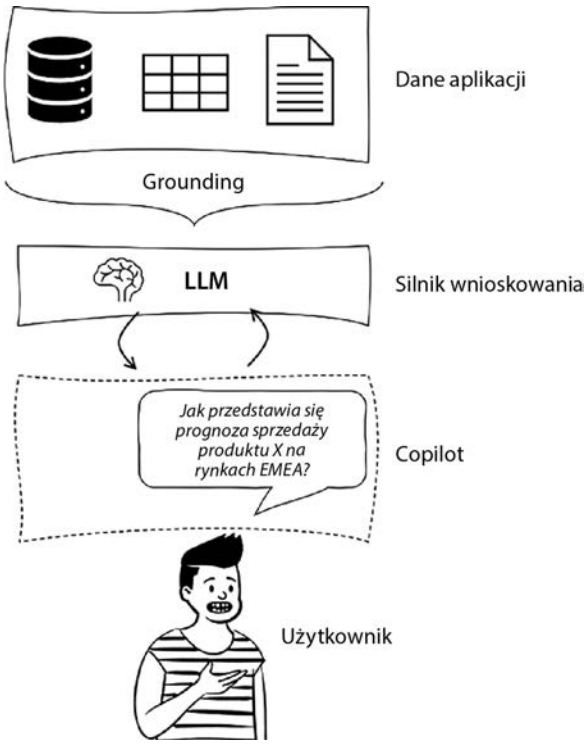
Uwaga

Na czym polega różnica między systemem copilot a frameworkiem typu RAG? RAG można postrzegać jako jeden ze wzorców architekturalnych, które kryją się za funkcjonalnością systemu copilot. Za każdym razem, gdy zachodzi potrzeba ograniczenia systemu do danych dotyczących konkretnej dziedziny, można skorzystać z frameworka typu RAG. Pamiętaj, że RAG to niejedyny wzorzec architekturalny stosowany przez funkcjonalność systemu copilot: używa się jeszcze innych frameworków, takich jak wywoływanie funkcji lub praca z wieloma agentami, które też przedstawiam w tej książce.

Na przykład przyjmujemy założenie o opracowaniu w firmie systemu copilot zapewniającego pracownikom możliwość korzystania z firmowej bazy wiedzy. Wprawdzie mogłoby to być zabawne, ale firma nie może dostarczać systemu, który pozwoliłby im zaplanować wakacje. (To oznaczałoby dla firmy konieczność udostępnienia użytkownikom narzędzia typu ChatGPT i pokrycia kosztów jego hostingu). Mamy do czynienia z sytuacją wręcz przeciwną: copilot ma być ograniczony do firmowej bazy wiedzy, aby mógł udzielać odpowiedzi na zapytania związane z konkretną dziedziną.

Przykładowy proces groundingu systemu copilot pokazałam na rysunku 2.3.

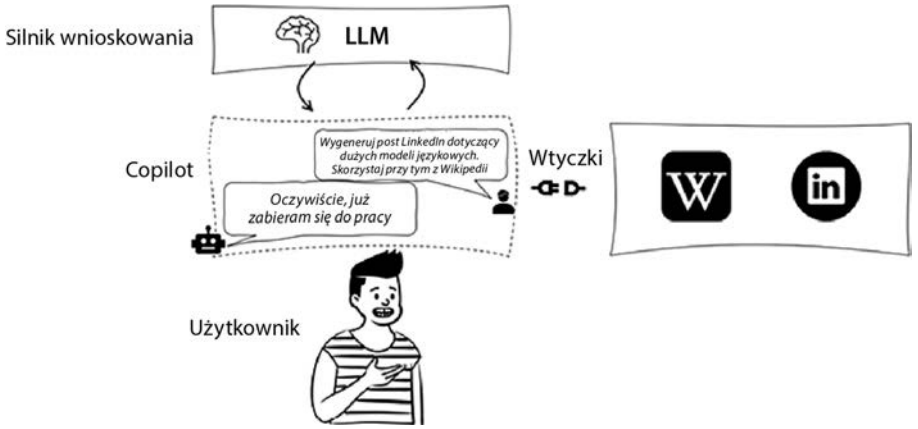
- **Możliwości systemu copilot mogą być rozbudowywane poprzez umiejętności**, które można zakodować bądź uzyskać za pomocą wywołań do innych modeli. W rzeczywistości duży model językowy (nasz silnik wnioskowania) może mieć dwa rodzaje ograniczeń.
  - **Ograniczona wiedza parametryczna.** Wynika ono z faktu, że baza wiedzy zawiera informacje zebrane do określonego dnia, co jest fizjologiczną cechą dużych modeli językowych. Tak naprawdę zbiór danych używany do trenowania dużego modelu językowego zawsze będzie „nieaktualny”,



Rysunek 2.3. Przykładowy proces groundingu systemu copilot

nie na czasie z bieżącymi trendami. Rozwiązaniem tego problemu może być wykorzystanie procesu groudingu w celu dodania wiedzy nieparametrycznej, co już wcześniej wyjaśniłam.

- **Brak władzy wykonawczej.** Oznacza ono, że duże modele językowe same z siebie nie mają uprawnień do przeprowadzania działań. Rozważ np. doskonale znany model ChatGPT: jeżeli poprosisz o wygenerowanie posta LinkedIn zawierającego podpowiedzi co do produktywności, treść posta musisz później skopiować i wkleić na swoim profilu LinkedIn, ponieważ ChatGPT nie ma takich uprawnień. To jest podstawowy powód, dla którego potrzebne są wtyczki. W tym kontekście wtyczka to komponent łączący duży model językowy ze światem zewnętrznym, który działa nie tylko w charakterze źródła danych wejściowych rozszerzającego nieparametryczną wiedzę dużego modelu językowego (np. wtyczka pozwala przeszukiwać zasoby internetu), ale również jako źródło danych wyjściowych, aby system copilot mógł faktycznie wykonywać akcje. Na przykład dzięki wtyczce LinkedIn nasz system wspomagany przez duży model językowy będzie mógł zarówno wygenerować post, jak też opublikować go w internecie. Spójrz na rysunek 2.4.



Rysunek 2.4. Przykład wtyczek dostępu do serwisów Wikipedii i LinkedIn

Zauważ, że zapytanie (prompt) użytkownika w języku naturalnym to nie są jedyne dane wejściowe przetwarzane przez model. W rzeczywistości to komponent o znaczeniu krytycznym logiki backendu naszej aplikacji wspomaganej przez duży model językowy i zestawu instrukcji dostarczanych modelowi. To *metazapytanie* (ang. *metaprompt*), czyli komunikat systemowy, jest przedmiotem nowej dyscypliny nazywanej **inżynierią zapytań** (ang. *prompt engineering*).

Definicja

Prompt engineering to proces pomagający projektować i optymalizować zapytania dla dużych modeli językowych pod kątem wielu różnych aplikacji i przedmiotów badań. Prompt jest krótkim fragmentem tekstu, który wskazuje dużemu modelowi językowemu rodzaj oczekiwanych danych wyjściowych. Umiejętności w zakresie inżynierii zapytań pomagają lepiej zrozumieć możliwości i ograniczenia dotyczące dużych modeli językowych.

Prompt engineering obejmuje wybór odpowiednich słów, wyrażeń, symboli i formatu, aby otrzymać oczekiwaną odpowiedź z dużego modelu językowego. Inżynieria zapytań wiąże się również z używaniem innych kontrolek, takich jak parametry, przykłady lub źródła danych, aby w ten sposób wpłynąć na sposób działania modelu. Na przykład, jeśli chcesz, aby wspomagana przez duży model językowy aplikacja wygenerowała odpowiedzi dla pięciolatka, możesz to wskazać w komunikacie systemowym w sposób podobny do tutaj przedstawionego: „Act as a teacher who explains complex concepts to 5-year-old children" (pol. *postaw się w roli nauczyciela, który próbuje wyjaśnić 5-latkowi skomplikowane koncepcje*).

W rzeczywistości Andrej Karpathy, wcześniej dyrektor działu sztucznej inteligencji w firmie Tesla, który w lutym 2023 roku powrócił do firmy OpenAI, stwierdził, że „angielski to jeden z obecnie najczęściej używanych języków programowania".

W rozdziale 4. bardzo dokładnie omówię koncepcję *prompt engineering*. Natomiast w następnym podrozdziale zamierzam skoncentrować się na pojawiających się frameworkach przeznaczonych do koordynowania sztucznej inteligencji.

Wprowadzenie do frameworków koordynowania sztucznej inteligencji umożliwiających osadzanie dużych modeli językowych w aplikacjach

Wcześniej w rozdziale wyjaśniłam, że istnieją dwa podstawowe aspekty, na które trzeba zwrócić uwagę podczas osadzania dużych modeli językowych w aplikacjach: techniczne i koncepcyjne. Wprawdzie aspekt koncepcyjny można wyjaśnić za pomocą zupełnie nowej kategorii oprogramowania nazywanej system copilot, ale w tym podrozdziale chcę pójść o krok dalej i pokazać, jak pod względem technicznym wygląda osadzanie i koordynowanie dużych modeli językowych w aplikacjach.

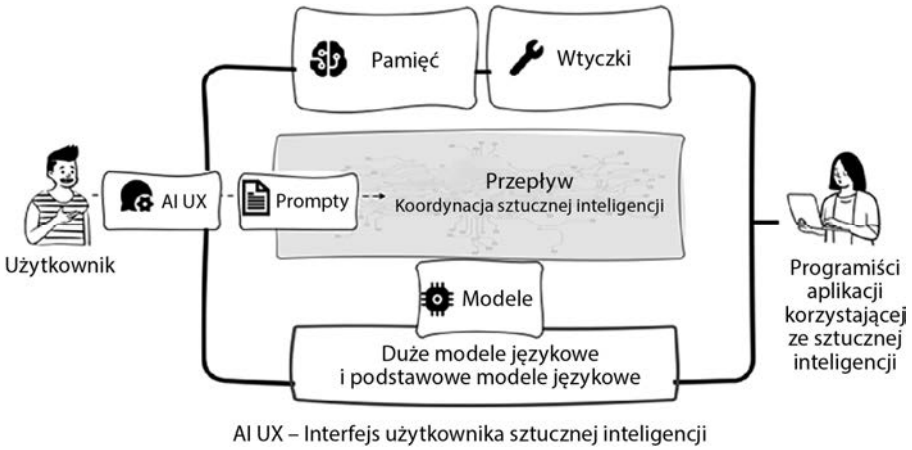
Najważniejsze komponenty frameworków koordynowania sztucznej inteligencji

Z jednej strony przesunięcie paradygmatu modeli podstawowych jest oznaką ogromnego uproszczenia w dziedzinie aplikacji wspomaganych przez sztuczną inteligencję: po generowaniu modeli obecnie trendem jest ich wykorzystywanie. Z drugiej jednak tworzeniu nowego rodzaju rozwiązań wspomaganych przez sztuczną inteligencję towarzyszy sporo problemów, ponieważ pojawiły się związane z dużymi modelami językowymi zupełnie nowe komponenty, którymi nigdy wcześniej nie trzeba było zarządzać w cyklu życiowym aplikacji. Na przykład mogą się zdarzać złośliwie działający użytkownicy, którzy spróbują zmienić instrukcje (czyli wspomniany wcześniej komunikat) dużego modelu językowego, aby aplikacja nie otrzymała poprawnych instrukcji. To jest przykład nowej kategorii zagrożeń, które okazują się powszechne w przypadku aplikacji wspomaganych przez duże modele językowe. Te zagrożenia trzeba eliminować za pomocą potężnych kontrataków bądź technik prewencyjnych.

Najważniejsze komponenty tego rodzaju aplikacji pokazałam na rysunku 2.5.

Warto znacznie dokładniej przeanalizować wymienione komponenty.

- **Modele.** To jest po prostu typ dużego modelu językowego, który ma zostać osadzony w aplikacji. Istnieją dwie podstawowe kategorie modeli.
  - **Własnościowe duże modele językowe.** Modele będące własnością konkretnych firm bądź organizacji. Przykładami są tutaj GPT-3 i GPT-4



Rysunek 2.5. Ogólna architektura aplikacji wykorzystującej duże modele językowe

opracowane przez OpenAI, a także model Bard opracowany przez Google. Skoro kod źródłowy i architektura tych modeli są niedostępne, nie można ich ponownie wytrenować na podstawie własnych danych, choć w razie potrzeby można je dostroić.

- **Otwartoźródłowe duże modele językowe.** Modele, których kod źródłowy i architektura są swobodnie dostępne i które tym samym można ponownie wytrenować na podstawie własnych danych. Przykładami są tutaj Falcon LLM opracowany przez *Technology Innovation Institute* (TII) w Abu Zabi, a także model Llama opracowany przez firmę Meta.

W następnym rozdziale znacznie dokładniej przedstawię dostępny obecnie zbiór dużych modeli językowych.

- **Pamięć.** Aplikacje wspomagane przez duże modele językowe używają interfejsu konwersacyjnego, który wymaga możliwości odwoływania się do wcześniejszych informacji pojawiających się w trakcie konwersacji. To można osiągnąć z zastosowaniem mechanizmu „pamięci", umożliwiającego aplikacji przechowywanie i pobieranie poprzednich interakcji. Warto w tym miejscu dodać, że te poprzednie interakcje stanowią dodatkową, nieparametryczną wiedzę dodawaną do modelu. Aby można było osiągnąć taki efekt, niezwykle ważne jest przechowywanie poprzednich konwersacji — poprawnie osadzonych — w wektorowej bazie danych VectorDB, czyli w podstawowym komponencie danych aplikacji.
- **Wtyczki.** Można je postrzegać jako dodatkowe moduły lub komponenty, które później można zintegrować z dużymi modelami językowymi, aby rozszerzyć ich funkcjonalność bądź przystosować je do wykonywania konkretnych zadań oraz do określonych zastosowań. Te wtyczki działają jako dodatki i wzbogacają możliwości dużych modeli językowych, które nie muszą ograniczać się do generowania języka i jego rozumienia.

Definicja

VectorDB to typ bazy danych przechowującej i pobierającej informacje na podstawie zwektoryzowanych danych, czyli reprezentacji liczbowej przechwytującej znaczenie tekstu i jego kontekst. Dzięki wykorzystaniu VectorDB można przeprowadzać semantyczne wyszukiwanie i pobieranie danych, uwzględniając przy tym podobieństwo znaczenia, zamiast używać słów kluczowych. Ta baza danych może również pomóc dużemu modelowi językowemu w wygenerowaniu trafniejszego i bardziej spójnego tekstu przez dostarczenie kontekstu i wzbogacenie wyników generowania. Przykładami wektorowych baz danych są Chroma, Elasticsearch, Milvus, Pinecone, Qdrant, Weaviate i **Facebook AI Similarity Search** (**FAISS**).

Bazę danych FAISS opracowano w firmie Facebook (obecnie Meta) w 2017 roku. Była wówczas jedną z pierwszych tego rodzaju baz danych. Miała zapewnić możliwość efektywnego wyszukiwania podobieństw oraz klastrowania wektorów zawierających ogromne ilości treści. Szczególnie przydatna okazuje się podczas pracy z dokumentami multimedialnymi oraz osadzonymi komponentami z obszerną treścią. Początkowo FAISS była wewnętrznym projektem badawczym firmy Facebook. Celem przyświecającym jej twórcom było lepsze wykorzystanie procesora graficznego w trakcie wyszukiwania podobieństw związanych z preferencjami użytkownika. Wraz z upływem czasu projekt ewoluował do postaci najszybszej dostępnej biblioteki przeznaczonej do wyszukiwania podobieństw. Ta biblioteka potrafi obsługiwać zbiory danych składające się z miliardów rekordów. Baza danych FAISS umożliwiła powstanie silników rekomendacji oraz systemów w postaci asystentów wspomaganych przez sztuczną inteligencję.

Dzięki użyciu wtyczki duży model językowy ma stać się bardziej wszechstronny i łatwiejszy do dostosowania przez programistów i użytkowników do własnych, jasno sprecyzowanych potrzeb. Wtyczki mogą być przeznaczone do wykonywania różnych zadań, a ponadto można je bezproblemowo zintegrować z architekturą dużych modeli językowych.

- **Prompty.** To prawdopodobnie najbardziej interesujący i najważniejszy komponent aplikacji wspomaganych przez duży model językowy. Nieco wcześniej w rozdziale przedstawiłam już stwierdzenie Andreja Karpathy'ego, że „angielski to jeden z obecnie najczęściej używanych języków programowania". W kolejnych rozdziałach przekonasz się, skąd wzięło się to stwierdzenie. Prompty można definiować na dwóch różnych poziomach.
  - **„Frontend", czyli to, co widzi użytkownik.** W takim przypadku „prompt" odwołuje się do danych wejściowych modelu. To sposób umożliwiający użytkownikowi pracę z aplikacją oraz zadawanie pytań w jego języku naturalnym.
  - **„Backend", czyli to, czego użytkownik nie widzi.** Język naturalny to niejedyny sposób umożliwiający użytkownikowi pracę z aplikacją przez użycie frontendu. To jednocześnie sposób, w jaki „programujemy" backend. W rzeczywistości za promptem użytkownika kryje się wiele instrukcji języka naturalnego, czyli metapromptów, przekazywanych modelowi,

aby mógł on sobie poradzić z udzieleniem odpowiedzi na pytanie użytkownika. Zadaniem metapromptów jest wskazanie modelowi, jak powinien działać. Na przykład, jeśli chcesz ograniczyć aplikację w taki sposób, aby udzielała odpowiedzi jedynie na pytania związane z dokumentacją dostarczoną w VectorDB, wówczas w metapromptach do modelu należy zamieścić następującą instrukcję „Answer only if the question is related to the provided documentation" (pol. *odpowiedz tylko wtedy, gdy pytanie wiąże się z dokumentacją*).

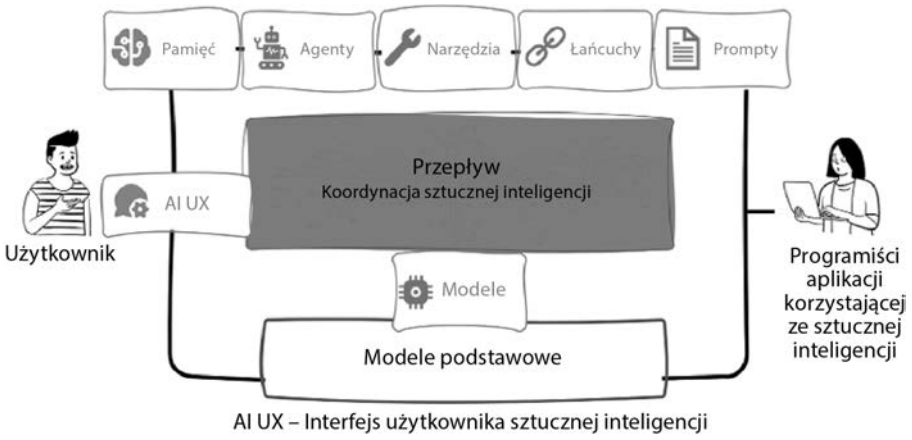
Wreszcie docieramy do sedna ogólnej architektury pokazanej na rysunku 2.5 we wcześniejszej części rozdziału, czyli do **frameworka koordynowania sztucznej inteligencji**. Dzięki takiej koordynacji można odwoływać się do lekkich bibliotek — wówczas znacznie łatwiejsze staje się osadzanie tych bibliotek w aplikacji oraz koordynowanie w niej dużych modeli językowych.

Gdy pod koniec 2022 roku duże modele językowe zyskały popularność, na rynku zaczęły się pojawiać frameworki. W kolejnych punktach skoncentruję się na trzech z nich: LangChain, Semantic Kernel i Haystack.

LangChain

LangChain to projekt otwartoźródłowy zainicjowany przez Harrisona Chase'a w październiku 2022 roku. Można go używać zarówno w Pythonie, jak i JS/TS. To framework umożliwiający opracowywanie aplikacji wspomaganych przez modele językowe. Takie aplikacje potrafią korzystać z danych (dzięki zastosowaniu groundingu) i agentów, co oznacza, że są w stanie współpracować ze środowiskami zewnętrznymi.

Kluczowe komponenty LangChain pokazałam na rysunku 2.6.



Rysunek 2.6. Komponenty frameworka LangChain

Można powiedzieć, że LangChain składa się z wymienionych tutaj modułów podstawowych.

- **Modele.** To są duże modele językowe lub modele podstawowe, które następnie będą używane przez aplikację. LangChain obsługuje modele własnościowe, takie jak dostępne w rozwiązaniach dostarczanych przez OpenAI i Azure OpenAI, a także modele otwartoźródłowe zamieszczone na platformie **Hugging Face Hub**.

Definicja

Hugging Face to firma (i społeczność) zajmująca się tworzeniem i udostępnianiem najnowszych modeli oraz narzędzi na potrzeby przetwarzania języka naturalnego i dla innych dziedzin związanych z uczeniem maszynowym. Ta firma opracowała Hugging Face Hub, czyli platformę, na której użytkownicy mogą tworzyć i odkrywać modele uczenia maszynowego, duże modele językowe, zbiory danych i prezentacje oraz współpracować nad nimi. Hugging Face Hub zawiera około 120 000 modeli, 20 000 zbiorów danych oraz 50 000 zadań i prezentacji z różnych dziedzin, takich jak obsługa dźwięku, obrazu i języka.

  Razem z modelami LangChain udostępnia również wiele związanych z promptami komponentów, dzięki którym zarządzanie przepływem promptów staje się łatwiejsze.

- **Konektory danych.** Odwołują się do elementów konstrukcyjnych niezbędnych do pobierania dodatkowej wiedzy zewnętrznej (np. w scenariuszach opartych na RAG), która ma zostać dostarczona modelowi. Przykładami konektorów danych są komponenty wczytujące dokumenty oraz modele osadzania tekstu.
- **Pamięć.** Pozwala aplikacji przechowywać odwołania do interakcji użytkownika, zarówno na krótką, jak i na dłuższą metę. Zwykle opiera się na zwektoryzowanych osadzeniach przechowywanych w bazie danych VectorDB.
- **Łańcuchy.** To wcześniej określone sekwencje akcji i wywołań do dużych modeli językowych, ułatwiające budowanie skomplikowanych aplikacji wymagających łączenia dużych modeli językowych ze sobą oraz z innymi komponentami. Przykład łańcucha może być następujący: pobierz zapytanie użytkownika, podziel je na mniejsze fragmenty, osadź je, wyszukaj podobne w bazie danych VectorDB, trzech najbardziej podobnych fragmentów w VectorDB użyj jako kontekstu do przedstawienia odpowiedzi oraz wygeneruj odpowiedź.
- **Agenty.** To encje kierujące podejmowaniem decyzji w aplikacjach wspomaganych przez duże modele językowe. Agenty mają dostęp do zbioru narzędzi oraz na podstawie danych wejściowych użytkownika i kontekstu mogą określać, które z nich zostanie użyte. Agenty są dynamiczne i elastyczne, a tym samym mogą zmieniać bądź dostosowywać działania w zależności od sytuacji lub celu.

LangChain ma następujące zalety:

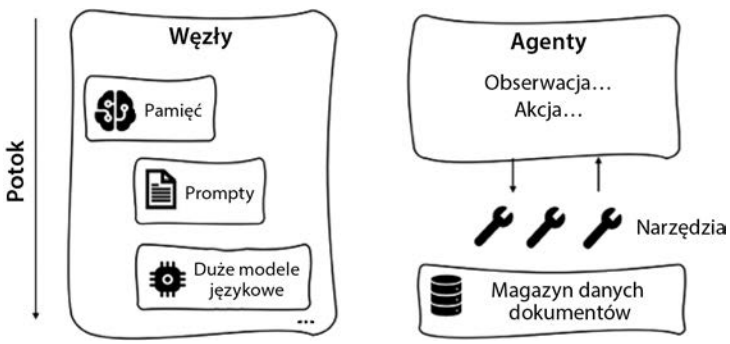
- Dla wspomnianych wcześniej komponentów LangChain dostarcza modułowe abstrakcje, niezbędne do pracy z modelami językowymi. Te abstrakcje są związane m.in. z promptami, pamięcią i wtyczkami.
- Razem z tymi komponentami LangChain oferuje również wcześniej przygotowane **łańcuchy**, które mają postać ustrukturyzowanych konkatenacji komponentów. Te łańcuchy mogą być wstępnie przygotowane do konkretnych przypadków użycia, a także można je dostosować do własnych potrzeb.

W drugiej części książki przedstawię serię aplikacji opartych na LangChain. Zatem począwszy od rozdziału 5. skoncentruję się na komponentach LangChain i ogólnych frameworkach.

Haystack

Haystack to oparty na Pythonie framework opracowany przez Deepset, czyli działający w Berlinie od 2018 roku startup, którego założycielami są Milos Rusic, Malte Pietsch i Timo Möller. Deepset zapewnia programistom narzędzia przeznaczone do tworzenia aplikacji korzystających z **przetwarzania języka naturalnego** (ang. *natural language processing*, NLP). Wraz z wprowadzeniem frameworka Haystack firma przeniosła te narzędzia na wyższy poziom.

Podstawowe komponenty Haystack pokazałam na rysunku 2.7.



Rysunek 2.7. Podstawowe komponenty frameworka Haystack

Zapoznaj się teraz znacznie dokładniej z tymi komponentami.

- **Węzły.** To komponenty przeznaczone do wykonywania konkretnych zadań lub funkcji. Przykładem może być pobieranie, odczytywanie, generowanie, tworzenie podsumowania itd. Węzłem może być duży model językowy bądź też inne narzędzie współdziałające z dużymi modelami językowymi lub innymi zasobami. W zakresie dużych modeli językowych Haystack obsługuje modele własnościowe, takie jak dostarczane przez OpenAI i Azure OpenAI, a także modele otwartoźródłowe pochodzące z platformy Hugging Face Hub.

- **Potoki.** To sekwencje wywołań do węzłów przeprowadzające działania związane z językiem naturalnym bądź współdziałające z innymi zasobami. Potok może wykonywać zapytanie albo indeksowanie, w zależności od tego, czy wykonywana jest operacja wyszukiwania w zbiorze dokumentów czy też operacja przygotowania dokumentów do wyszukiwania. Potoki są z góry określone i zdefiniowane na stałe, a tym samym nie zmieniają się i nie są dostosowywane na podstawie kontekstu ani danych wejściowych użytkownika.
- **Agent.** To jest encja używająca dużych modeli językowych do wygenerowania właściwych odpowiedzi na skomplikowane zapytania. Agent ma dostęp do zbioru narzędzi, którymi mogą być potoki lub węzły, a także może decydować, które narzędzie zostanie użyte, na podstawie kontekstu bądź danych wejściowych użytkownika. Agent jest dynamiczny i elastyczny, a tym samym może zmieniać lub dostosowywać działania w zależności od sytuacji albo celu.
- **Narzędzia.** To funkcje, które agent może wywoływać w celu wykonywania zadań związanych z językiem naturalnym bądź współdziałania z innymi zasobami. Narzędziami mogą być potoki lub węzły dostępne dla agenta. Można je również grupować w zestawy narzędzi, czyli w zbiory narzędzi przeznaczone do wykonywania określonych zadań.
- **Magazyn danych dokumentów.** To backend przechowujący dokumenty i obsługujący ich wyszukiwanie. Magazyn danych może być zbudowany na podstawie jednej z wielu dostępnych technologii, obejmującej także rozwiązania w postaci wektorowych baz danych (np. FAISS, Milvus lub Elasticsearch).

Oto wybrane korzyści z używania frameworka Haystack.

- **Łatwość użycia.** Framework Haystack jest prosty i przyjazny użytkownikowi. Często wybiera się go do mniejszych zadań i szybkiego tworzenia prototypów.
- **Jakość dokumentacji.** Przygotowaną dla frameworka dokumentację uważa się za wysokiej jakości. Pomaga ona programistom w tworzeniu systemów wyszukiwania, udzielania odpowiedzi i tworzenia podsumowań oraz rozwiązań z zakresu konwersacyjnej sztucznej inteligencji.
- **Pełny framework.** Haystack obsługuje cały cykl życiowy projektu dużego modelu językowego, od wstępnego przetwarzania danych aż po wdrożenie modelu. Idealnie sprawdza się w systemach wyszukiwania na ogromną skalę oraz w systemach pobierania informacji.
- Inną użyteczną cechą omawianego frameworka jest możliwość wdrożenia jako API REST, co umożliwia bezpośrednie używanie go.

Semantic Kernel

Semantic Kernel to trzecie otwartoźródłowe SDK, które przedstawię w tym rozdziale. Opracowała je firma Microsoft. Początkowo było dostępne w języku C#, a obecnie — również w Pythonie.

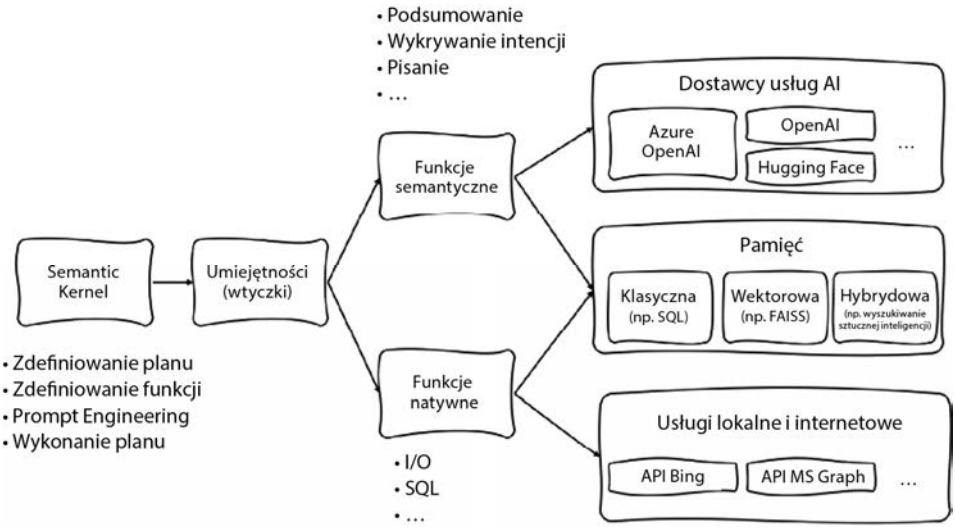
Nazwa tego frameworka nawiązuje do koncepcji jądra (ang. *kernel*), które ogólnie rzecz biorąc, odwołuje się do podstawy, inaczej rdzenia, systemu. W kontekście omawianego frameworka jądro jest przeznaczone do działania w charakterze silnika, który będzie obsługiwał dane wejściowe użytkownika przez łączenie serii komponentów i ich konkatenację na postać potoków, co sprzyja tzw. **złożeniu funkcji**.

Definicja

W matematyce złożenie funkcji oznacza połączenie dwóch funkcji ze sobą, którego celem jest powstanie nowej funkcji. Idea polega na użyciu danych wyjściowych jednej z funkcji jako danych wejściowych drugiej, w czego wyniku powstaje łańcuch funkcji. Złożenie dwóch funkcji f i g jest zapisywane w postaci $(f \circ g)$, gdzie funkcja g będzie zastosowana jako pierwsza, a po niej zostanie użyta $f \rightarrow (f \circ g)(x) = f(g(x))$.

W informatyce złożenie funkcji to potężna koncepcja, dzięki której możliwe jest tworzenie znacznie bardziej zaawansowanego kodu wielokrotnego użycia, co odbywa się przez łączenie mniejszych funkcji w większe. W ten sposób usprawnia się modułowość i organizację kodu, a tworzone programy stają się łatwiejsze do odczytu i późniejszej obsługi technicznej.

Anatomię frameworka Semantic Kernel pokazałam na rysunku 2.8.



Rysunek 2.8. Anatomia frameworka Semantic Kernel

Framework Semantic Kernel oferuje następujące komponenty główne:

- **Modele.** To są duże modele językowe lub modele podstawowe, które następnie będą używane przez aplikację. Semantic Kernel obsługuje modele własnościowe, takie jak dostarczane przez OpenAI i Azure OpenAI, a także modele otwartoźródłowe pochodzące z platformy Hugging Face Hub.

- **Pamięć.** Pozwala aplikacji przechowywać odwołania do interakcji użytkownika, zarówno na krótką, jak i na dłuższą metę. W przypadku frameworka Semantic Kernel dostęp do pamięci odbywa się na jeden z trzech sposobów.
  - **Pary klucz-wartość.** Polega na zapisywaniu zmiennych środowiskowych, które przechowują proste informacje, takie jak imiona lub daty.
  - **Lokalna pamięć masowa.** Polega na zapisywaniu informacji w pliku, który może być później pobrany na podstawie jego nazwy. To może być plik w formacie np. CSV lub JSON.
  - **Wyszukiwanie pamięci masowej frameworka Semantic.** To rozwiązanie bardzo podobne do użycia pamięci we frameworkach LangChain i Haystack, ponieważ korzysta z osadzonych elementów do przedstawienia i wyszukiwania informacji tekstowych na podstawie ich znaczenia.
- **Funkcje.** Można je postrzegać jako umiejętności łączące w sobie duże modele językowe i kod. Dzięki temu zapytanie użytkownika ma być zrozumiałe, aby możliwe było udzielenie na nie odpowiedzi. Mamy dwa rodzaje funkcji.
  - **Funkcje semantyczne.** To rodzaj szablonu promptu, który jest zapytaniem języka naturalnego oraz określa format danych wejściowych i wyjściowych dla dużych modeli językowych. Ponadto obejmuje konfigurację promptu, która określa parametry dla dużego modelu językowego.
  - **Funkcje natywne.** Odwołują się do natywnego kodu komputera, który może przekazywać intencję przechwyconą przez funkcję semantyczną i wykonać związane z nią zadanie.

  Na przykład funkcja semantyczna może nakazać dużemu modelowi językowemu wygenerowanie krótkiego akapitu dotyczącego sztucznej inteligencji, podczas gdy funkcja natywna mogłaby faktycznie opublikować ten tekst w mediach społecznościowych, np. w serwisie LinkedIn.
- **Wtyczki.** To konektory przeznaczone dla źródeł zewnętrznych lub systemów, które mają dostarczać informacji dodatkowych bądź umożliwiać przeprowadzanie autonomicznych działań. Framework Semantic Kernel standardowo oferuje zestaw wtyczek, np. Microsoft Graph Connectors. Istnieje również możliwość samodzielnego tworzenia wtyczek przez wykorzystanie funkcji (natywnych, semantycznych bądź połączenia obu tych rodzajów).
- **Planer.** Skoro duże modele językowe mogą być postrzegane jako silniki wnioskowania, zatem można je wykorzystać do automatycznego tworzenia łańcuchów bądź potoków, aby w ten sposób spełnić nowe wymagania użytkowników. Ułatwia to planer, czyli funkcja pobierająca dane wejściowe w postaci zadania określonego przez użytkownika i generująca zbiór akcji, wtyczek i innych funkcji niezbędnych do osiągnięcia wyznaczonego celu.

Oto wybrane korzyści wynikające z używania frameworka Semantic Kernel.

- **Lekki framework obsługujący język C#.** Semantic Kernel to niewielki framework, który oferuje obsługę języka programowania C#. Jest więc

doskonałym wyborem dla programistów C# bądź programistów używających frameworka .NET.

- **Szeroki zakres przypadków użycia.** Semantic Kernel to wszechstronny framework, który obsługuje wiele różnych zadań związanych z dużymi modelami językowymi.
- **Sprawdzony przez branżę.** Semantic Kernel to framework opracowany przez firmę Microsoft, która użyła go do zbudowania własnych systemów typu copilot. Dlatego też można stwierdzić, że ten framework uwzględnia potrzeby branży i jest solidnym narzędziem przeznaczonym dla aplikacji działających na skalę korporacji.

Jak wybrać framework?

Wszystkie trzy omówione pokrótce frameworki oferują mniej więcej podobne komponenty podstawowe, czasami nazywane w odmienny sposób, i zapewniają obsługę wszystkich bloków przedstawionych w koncepcji systemu copilot. Tak więc naturalnym pytaniem może być: „Który z tych frameworków najlepiej jest wybrać podczas tworzenia aplikacji wspomaganej przez duży model językowy?". Na tak zadane pytanie nie ma dobrej ani złej odpowiedzi, wszystkie są niezwykle trafne. Jednak istnieją pewne funkcjonalności, które mogą mieć większe znaczenie w konkretnych przypadkach użycia bądź ze względu na preferencje programistów. Zapoznaj się z wybranymi kryteriami, które warto rozważyć.

- **Język programowania, w którym czujesz się komfortowo lub którego użycie preferujesz.** Poszczególne frameworki mogą obsługiwać odmienne języki programowania lub oferują różne poziomy zgodności bądź integracji z nimi. Na przykład Semantic Kernel obsługuje języki C#, Python i Java, podczas gdy LangChain i Haystack opierają się przede wszystkim na Pythonie (nawet pomimo tego, że we frameworku LangChain pojawiła się obsługa języków JS/TS). Możesz zdecydować się na framework dopasowany do swoich obecnych możliwości lub preferencji bądź też framework, który pozwoli używać języka najbardziej odpowiedniego do dziedziny aplikacji albo środowiska.
- **Rodzaj i złożoność zadań języka naturalnego, które mają być wykonywane lub obsługiwane.** Poszczególne frameworki mogą mieć różne możliwości lub odmienny zestaw funkcji przeznaczonych do obsługi różnych zadań języka naturalnego, takich jak tworzenie podsumowań, generowanie tekstu, tłumaczenie tekstu, wnioskowanie itd. Na przykład LangChain i Haystack udostępniają narzędzia i komponenty przeznaczone do koordynowania i wykonywania zadań języka naturalnego. Z kolei Semantic Kernel pozwala używać funkcji semantycznych języka naturalnego do wywoływania dużych modeli językowych i usług. Prawdopodobnie zdecydujesz się na framework oferujący oczekiwaną funkcjonalność i elastyczność, być może niezbędne do osiągnięcia celów aplikacji lub realizacji konkretnych scenariuszy.

- **Możliwości dostosowania do własnych potrzeb i poziom kontroli, jakich oczekuje się od dużych modeli językowych, w tym ich parametrów i opcji, lub jakie są niezbędne w danym zastosowaniu.** Poszczególne frameworki mogą mieć odmienne sposoby na to, aby uzyskać dostęp, przeprowadzić konfigurację bądź też dostroić duży model językowy lub jego parametry i opcje, takie jak wybór modelu, projekt zapytania, szybkość działania, format danych wyjściowych itd. Na przykład Semantic Kernel zapewnia konektory, które ułatwiają dodawanie pamięci i modeli do aplikacji wspomaganych przez sztuczną inteligencję, podczas gdy LangChain i Haystack umożliwiają dołączanie różnych komponentów odpowiedzialnych za obsługę magazynu danych dokumentów, a także za pobieranie, odczytywanie, generowanie danych, tworzenie podsumowań i ocenę informacji. Prawdopodobnie zdecydujesz się na framework zapewniający oczekiwany lub niezbędny poziom kontroli i dostosowania do własnych potrzeb dużych modeli językowych, w tym ich parametrów i opcji.
- **Dostępność i jakość dokumentacji, samouczków i przykładów oraz wsparcie, jakie zapewnia jego społeczność.** Poszczególne frameworki mogą charakteryzować się różnymi poziomami dokumentacji, samouczków, przykładów i wsparcia ze strony społeczności, co z kolei mogłoby pomóc w poznaniu frameworka oraz jego użyciu, a także rozwiązywaniu ewentualnych problemów. Na przykład framework Semantic Kernel oferuje witrynę internetową razem z dokumentacją, samouczkami, przykładami i kanałem Discord przeznaczonym dla społeczności. W przypadku frameworka LangChain mamy repozytorium GitHub razem z dokumentacją, przykładami i możliwością zgłaszania błędów. Z kolei framework Haystack oferuje witrynę internetową wraz z dokumentacją, samouczkami, prezentacjami, postami na blogu i kanałem Slack przeznaczonym dla społeczności. Prawdopodobnie zdecydujesz się na framework oferujący dobrej jakości dokumentację, samouczki, przykłady i wsparcie ze strony społeczności, ponieważ dzięki temu łatwiej będzie rozpocząć pracę z danym frameworkiem oraz rozwiązywać pojawiające się problemy.

W tabeli 2.1 przedstawiłam krótkie podsumowanie różnic między trzema frameworkami omówionymi w rozdziale.

Tabela 2.1. Porównanie trzech frameworków koordynowania sztucznej inteligencji

Funkcjonalność	LangChain	Haystack	Semantic Kernel
Obsługa dużych modeli językowych	Własnościowe i otwartoźródłowe	Własnościowe i otwartoźródłowe	Własnościowe i otwartoźródłowe
Obsługiwane języki programowania	Python i JS/TS	Python	C#, Java i Python
Koordynacja procesu	Łańcuchy	Potoki węzłów	Potoki funkcji
Wdrożenie	Brak API REST	API REST	Brak API REST
Funkcjonalność	**LangChain**	**Haystack**	**Semantic Kernel**

Ogólnie rzecz biorąc, wszystkie omówione w rozdziale frameworki oferują szeroki zakres narzędzi, a także umożliwiają ich integrowanie z aplikacjami wspomaganymi przez duże modele językowe. Najrozsądniejszym podejściem będzie wybór tego frameworka, który najlepiej odpowiada Twoim obecnym umiejętnościom bądź jest najbardziej zgodny z ogólnym podejściem stosowanym przez firmę.

Podsumowanie

W rozdziale dokładniej przedstawiłam temat nowych sposobów na tworzenie aplikacji wspomaganych przez duże modele językowe. Ponadto wprowadziłam koncepcję systemu copilot i omówiłam nowe rozwiązania w zakresie koordynowania sztucznej inteligencji. Skoncentrowałam się przy tym na trzech projektach — LangChain, Haystack i Semantic Kernel. Omówiłam funkcjonalności, jakie zapewniają, najważniejsze komponenty oraz wybrane kryteria, które warto wziąć pod uwagę podczas wybierania frameworka.

Gdy już wybierzesz framework koordynowania sztucznej inteligencji, następnym ważnym krokiem będzie wybór dużego modelu językowego, który będzie osadzony w aplikacji. W następnym rozdziale przedstawię więc najważniejsze z takich modeli, które są obecnie dostępne. To będą modele zarówno własnościowe, jak i otwartoźródłowe. Wyjaśnię także wybrane kryteria związane z wyborem odpowiednich modeli, z uwzględnieniem przypadków użycia aplikacji.

Odwołania

- Repozytorium LangChain: *https://github.com/langchain-ai/langchain*
- Dokumentacja frameworka Semantic Kernel: *https://learn.microsoft.com/en-us/semantic-kernel/get-started/supported-languages*
- Stos systemu copilot: *https://devblogs.microsoft.com/microsoft365dev/a-pro-code-guide-to-build-your-custom-copilot-faster/*
- Wideo *The Copilot system*: *https://youtu.be/E5g20qmeKpg*

Odkryj, jak łatwo model generatywnej AI zintegruje się z Twoją aplikacją!

Duże modele językowe (**LLM**) stały się technologicznym przełomem. Ich wszechstronność i funkcjonalność sprawiły, że coraz częściej mówi się o nowej erze inteligentnie działających urządzeń i aplikacji. Umiejętność zastosowania **LLM** we własnych projektach już dziś jest koniecznością dla wielu projektantów i programistów.

Dzięki tej książce opanujesz podstawowe koncepcje związane z użyciem **LLM**. Poznasz unikatowe cechy i mocne strony kilku najważniejszych modeli (w tym GPT, Gemini, Falcon). Następnie dowiesz się, w jaki sposób LangChain, lekki framework Pythona, pozwala na projektowanie inteligentnych agentów do przetwarzania danych o nieuporządkowanej strukturze. Znajdziesz tu również informacje na temat dużych modeli podstawowych, które wykraczają poza obsługę języka i potrafią wykonywać różne zadania związane na przykład z grafiką i dźwiękiem. Na koniec zgłębisz zagadnienia dotyczące ryzyka związanego z **LLM**, a także poznasz techniki uniemożliwiania tym modelom potencjalnie szkodliwych działań w aplikacji.

W książce:

- architektura dużych modeli językowych
- unikatowe funkcje **LLM**
- komponenty służące do koordynacji sztucznej inteligencji, w tym tworzenia frontendu
- użycie wiedzy nieparametrycznej i wektorowych baz danych
- dostrajanie dużych modeli językowych do własnych potrzeb
- odpowiedzialność i etyka w systemach korzystających z **LLM**

Valentina Alto pracuje w firmie Microsoft, gdzie od 2022 roku zajmuje się wdrożeniami AI w branży produkcyjnej i farmaceutycznej. Specjalizuje się w inżynierii nowoczesnych platform i frameworków danych, IoT, Azure Machine Learning i Azure Cognitive Services. Tworzy też rozwiązania analityczne i pulpity nawigacyjne w Power BI. Jest autorką książek i wielu artykułów technicznych.

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 978-83-289-1894-8
9 788328 918948
Cena: 89,00 zł
packt