OKIEM EKSPERTA

Generatywna sztuczna inteligencja z LangChain

Budowanie aplikacji AI opartych na LLM z użyciem Pythona, ChatGPT i innych modeli językowych

Ben Auffarth

Helion

packt

Tytuł oryginału: Generative AI with LangChain: Build large language model (LLM) apps with Python, ChatGPT and other LLMs

Tłumaczenie: Anna Mizerska

ISBN: 978-83-289-1436-0

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/genesz
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: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

Rozdział 2

LangChain dla aplikacji opartych na modelach LLM

Duże modele językowe (**LLM**, ang. *Large Language Models*) takie jak model GPT-4 pokazały niesamowite możliwości w generowaniu tekstu, który trudno odróżnić od tego napisanego przez człowieka. Jednak korzystanie z modeli LLM poprzez API ma swoje ograniczenia. Dlatego zamiast tego można używać API w połączeniu z innymi źródłami danych i narzędziami, by tworzyć jeszcze potężniejsze aplikacje. W niniejszym rozdziale wprowadzimy framework LangChain, za pomocą którego możemy zniwelować ograniczenia modeli LLM i budować innowacyjne aplikacje oparte na modelach językowych. W tym rozdziale chcę Ci pokazać potencjał, jaki tkwi w połączeniu nowinek ze świata sztucznej inteligencji z solidnym frameworkiem, czyli LangChain.

Na początek zostaną wymienione ograniczenia, z którymi trzeba się mierzyć, gdy chcemy używać samych modeli LLM. Do tych ograniczeń możemy zaliczyć między innymi brak zewnętrznej bazy wiedzy, niepoprawne rozumowanie i niemożność podjęcia działania. LangChain zapewnia rozwiązania tych problemów poprzez integrację różnych dostępnych od ręki komponentów przeznaczonych do konkretnych działań. Zobaczymy, jak programiści mogą wykorzystywać możliwości frameworku LangChain w celu tworzenia niestandardowych rozwiązań do przetwarzania języka naturalnego oraz opowiemy o komponentach i koncepcjach leżących u podstaw LangChain.

Ten rozdział ma pokazać, jak LangChain umożliwia budowanie dynamicznych, innowacyjnych, związanych z danymi aplikacji przez zwykłe żądania wysyłane do API modeli LLM. Na koniec omówimy najważniejsze koncepcje związane z frameworkiem LangChain, między innymi łańcuchy, generowanie planu działania i pamięć, które pomogą nam zrozumieć sposób działania tego frameworku.

Ten rozdział składa się z następujących części:

- Wykraczanie poza papugę stochastyczną.
- Co to jest LangChain?
- Kluczowe komponenty LangChain.
- Jak działa LangChain?
- Porównanie LangChain z innymi frameworkami.

Wykraczanie poza papugę stochastyczną

Modele LLM zwróciły na siebie uwagę i zyskały popularność dzięki umiejętności generowania tekstu takiego jak człowiek, a co więcej, są w stanie rozumieć język naturalny, dzięki czemu są przydatne w przypadkach związanych z generowaniem zawartości, klasyfikacją tekstu i podsumowywaniem. Jednak płynność, jaką wykazują ostatnio duże modele językowe, przesłania poważne defekty uniemożliwiające pewne rzeczywiste zastosowania. Koncepcja papugi stochastycznej pomaga naświetlić ten podstawowy problem.

Papugi stochastyczne to modele LLM, które wytwarzają przekonująco brzmiący tekst, ale bez prawdziwego zrozumienia słów. Ten termin został ukuty przez Emily Bender, Timnita Gebru, Margaret Mitchell oraz Angelinę McMillan-Major i pierwszy raz ukazał się we wpływowym artykule naukowym *On the Dangers of Stochastic Parrots* (2021). Papuga stochastyczna jest krytyką modeli, które bezmyślnie naśladują wzorce językowe. Bez osadzenia w rzeczywistym świecie modele mogą odpowiadać niepoprawnie, bez związku z pytaniem, nieetycznie lub bez większego sensu logicznego.

Zwiększenie zestawu danych i mocy obliczeniowej nie wpływa na możliwości rozumowania czy na zdrowy rozsądek. Modele LLM borykają się z wyzwaniami takimi jak luki kompozycyjne (*Measuring and Narrowing the Compositionality Gap in Language Models*, Ofir Press i współpracownicy, 2023). To znaczy, że modele LLM nie mogą łączyć interfejsów ani przystosowywać odpowiedzi do nowych sytuacji. Aby pokonać te przeszkody, należy wdrożyć zaawansowane techniki, które dadzą możliwość modelom LLM naprawdę rozumieć tekst. Samo skalowanie modeli nie zmieni papugi stochastycznej w przynoszący korzyści system. Tutaj potrzebna jest edukacja modeli poprzez monity, tok myślowy, osadzenie w rzeczywistości i inne techniki.

Przyjrzyjmy się temu problemowi bliżej. Jeśli nie interesują Cię te szczegóły, możesz przejść do następnego podrozdziału. Teraz skupimy się na ograniczeniach modeli LLM, zastanowimy się, jak pokonać te ograniczenia, i dowiemy się, jak LangChain ułatwia aplikacjom systematyczne zmniejszanie negatywnych skutków tych ograniczeń i rozbudowuje funkcjonalność modeli LLM.

Ograniczenia modeli LLM

Jak już wspomniano wcześniej, modele LLM oferują imponujące możliwości, ale cierpią z powodu ograniczeń, które utrudniają uzyskanie odpowiedniej wydajności w pewnych przypadkach użycia. Trzeba rozumieć te ograniczenia, by móc skutecznie tworzyć aplikacje. Ograniczeniami modeli LLM są między innymi:

- **Nieaktualna wiedza**. Modele LLM polegają jedynie na danych treningowych. Bez integracji z zewnętrznym źródłem danych nie mogą dostarczać aktualnych informacji ze świata rzeczywistego.
- **Niemożność podjęcia działania**. Modele LLM nie mogą wykonywać zapętlonych akcji, takich jak wyszukiwanie, obliczenia czy sprawdzanie, co znacząco wpływa na ich funkcjonalność.

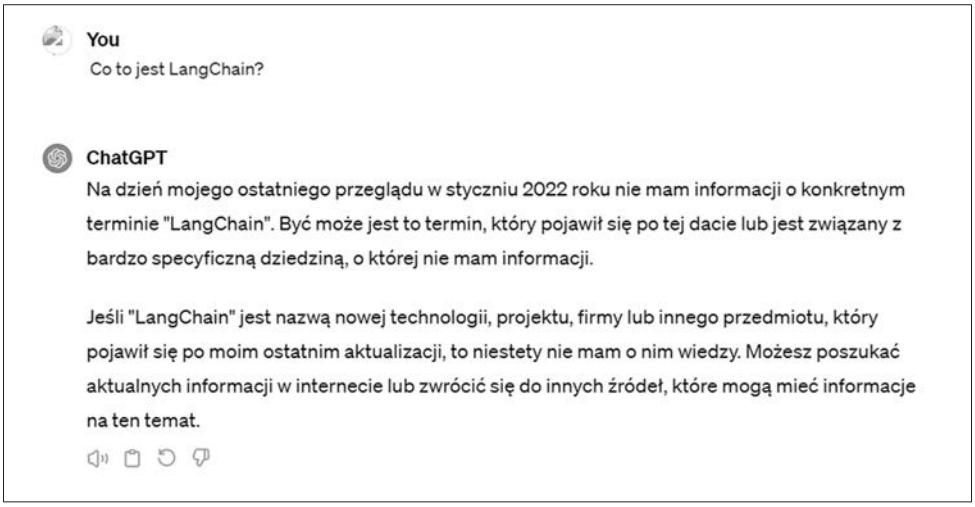
- **Brak kontekstu**. Modele LLM mają problem z przyjmowaniem odpowiedniego kontekstu, pochodzącego między innymi z poprzednich rozmów lub przyjmowaniem dodatkowych szczegółów wymaganych do podawania spójnych i przydatnych odpowiedzi.
- **Ryzyko halucynacji**. Niewystarczająca wiedza na pewne tematy może prowadzić do generowania niepoprawnej i pozbawionej sensu zawartości, jeśli model LLM nie jest odpowiednio osadzony w rzeczywistości.
- **Stereotypy i dyskryminacja**. W zależności od danych, na jakich były trenowane, modele LLM mogą przejawiać tendencyjność, przez co mogą mieć podłoże religijne, być przesiąknięte daną ideologią lub z natury polityczne.
- **Brak przejrzystości**. Działanie bardzo dużych, złożonych modeli może nie być jasne, przez co może być trudne do zrozumienia, a w konsekwencji zgodność z wartościami ludzi może być niełatwa do osiągnięcia.
- **Brak kontekstu**. Modele LLM mogą mieć problem ze zrozumieniem i przyswojeniem kontekstu z poprzednich monitów czy rozmów. Mogą nie pamiętać wcześniej podanych szczegółów lub nie będą w stanie podać dodatkowych informacji poza tym, co zostało podane w monitach.

Warto przytoczyć przykłady tych ograniczeń, gdyż to bardzo ważna kwestia. Jak już wspomniano wcześniej, modele LLM mierzą się ze znacznymi ograniczeniami związanymi z brakiem dostępu do najbardziej aktualnej wiedzy oraz nie mogą samodzielnie podejmować akcji, co ogranicza ich skuteczność w wielu rzeczywistych scenariuszach. Na przykład modele LLM nie mają naturalnych połączeń z zewnętrznymi źródłami informacji. Modele są ograniczone do danych treningowych, które dość szybko się dezaktualizują. Modele LLM nie są świadome wydarzeń, które nastąpiły po treningu. Dlatego jeśli zapytasz model LLM o najnowsze wiadomości, to nie będzie w stanie podać odpowiedzi bez zewnętrznego osadzenia.

Co więcej, modele LLM nie mogą dynamiczne wchodzić w interakcje z otaczającym je światem. Nie mogą sprawdzać pogody, wyszukiwać danych lokalnych i nie mają dostępu do dokumentów. Bez możliwości wyszukiwania informacji w internecie, komunikacji poprzez API, wykonywania obliczeń i podejmowania działań na podstawie nowych monitów modele LLM działają tylko w obrębie informacji, na których były szkolone. Nawet gdy rozmawiamy z modelem LLM na temat ujęty w danych treningowych, będzie miał on problem z objęciem bieżącego kontekstu bez dostępu do zewnętrznej bazy wiedzy. Na przykład model LLM może płynnie debatować o regułach makroekonomii stosowanych w analizie finansowej, ale nie uda mu się przeprowadzić analizy z uwzględnieniem bieżących danych i zastosowaniem odpowiednich obliczeń statystycznych. Bez możliwości dynamicznego wyszukiwania nasza rozmowa z modelem LLM o finansach będzie ogólna i teoretyczna. Modele LLM mogą również bardzo elokwentnie opisywać przeszłe wydarzenia w danej dziedzinie, ale nie są w stanie podać najnowszych informacji ani bieżącego stanu wiedzy na ten sam temat.

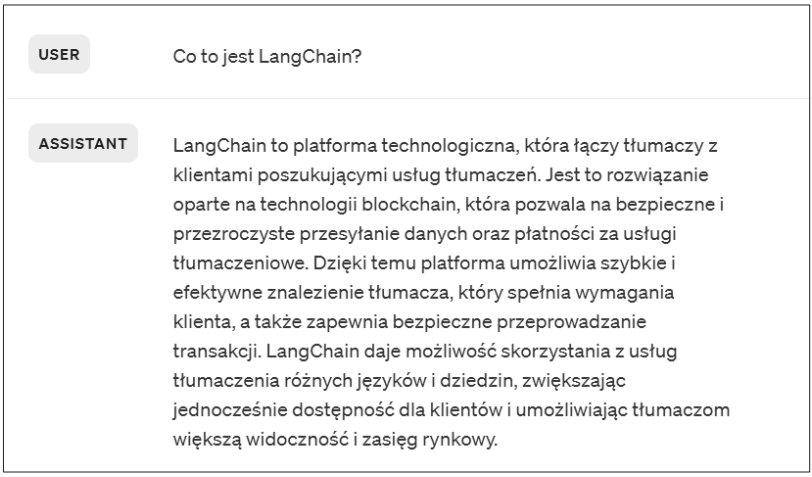
Te ograniczenia można pokonać za pomocą architektur łączących w sobie zewnętrzne źródła danych, programy analityczne i różne narzędzia. Odizolowane modele LLM nie mają łączności z rzeczywistością, co w wielu zastosowaniach jest niezbędne. Niesamowite umiejętności generowania języka naturalnego muszą iść w parze z odpowiednim osadzeniem w rzeczywistości i podejmowaniem właściwych działań w celu wysnuwania merytorycznych wniosków. Bez tego model może zwracać elokwentny, ale bezwartościowy tekst.

Zobaczmy kilka przykładowych problemów z modelami LLM. Problem nieaktualnych danych widać na rysunku 2.1, na którym pytamy ChatGPT firmy OpenAI o LangChain.



Rysunek 2.1. ChatGPT — problem braku najnowszych informacji

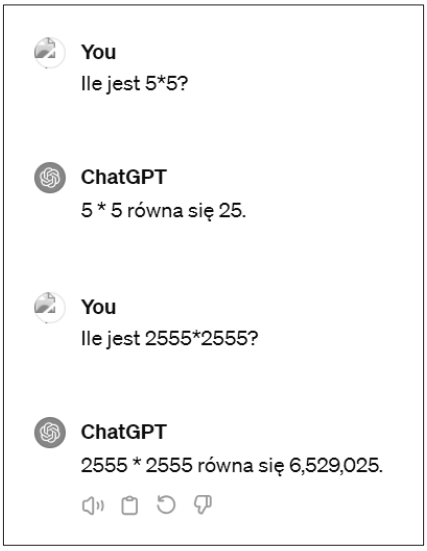
W tym przypadku, choć model niewiele nam pomógł, poprawnie zrozumiał nasze pytanie i podał odpowiednią informację zwrotną, co jednak nie zawsze ma miejsce. Jeśli dostaniesz się do modelu przez inne punkty końcowe lub użyjesz innych modeli, to możesz uzyskać zmyślone informacje (halucynacje). Ponadto model może nie mieć wiedzy na temat pewnych kwestii lub może odwoływać się do zupełnie innych zagadnień. Po zadaniu tego samego pytania w OpenAI Playground otrzymałem odpowiedź widoczną na rysunku 2.2.



Rysunek 2.2. OpenAI Playground z modelem GPT-3.5

W tym przypadku widzimy, że model mówi o innym narzędziu LangChain, które według dostarczonej odpowiedzi jest platformą opartą na technologii blockchain przeznaczoną dla tłumaczy. Mamy tutaj problem z brakiem związku z tematem, co można określić mianem halucynacji. Można to rozwiązać przez dostęp do zewnętrznych danych, między innymi do API serwisów pogodowych, preferencji użytkownika lub do informacji z internetu. Dzięki temu będzie można tworzyć spersonalizowane i dokładne aplikacje pracujące z naturalnym językiem.

Modele LLM mają problem z rozwiązywaniem zadań wymagających logicznego rozumowania i z zadaniami matematycznymi. Nawet zaawansowane modele LLM słabo radzą sobie z zagadnieniami matematycznymi na poziomie szkoły średniej i nie są w stanie wykonać prostych obliczeń matematycznych, jeśli wcześniej ich nie widziały. Rysunek 2.3 przedstawia ten problem.



Rysunek 2.3. Obliczenia matematyczne w wykonaniu ChatGPT

Jak widać, model poprawnie odpowiedział na pierwsze pytanie, ale nie udało mu się poprawnie obliczyć drugiego zadania. Jeśli zastanawiasz się, jaki jest prawidłowy wynik drugiego mnożenia, to spójrz na rysunek 2.4, na którym widać wynik zwracany przez kalkulator.

```
(base) ~ % bc -l
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
2555 * 2555
6528025
```

Rysunek 2.4. Mnożenie z użyciem kalkulatora (uniksowy kalkulator dowolnej precyzji)

Model LLM nie zapisał wcześniej wyniku obliczenia lub po prostu nie widział go w danych treningowych wystarczającą liczbę razy, by prawidłowo go zapamiętać poprzez przypisanie wag. Dlatego nie potrafi podać prawidłowego rozwiązania. Model LLM nie jest dobrym narzędziem do tego typu zadań.

Wdrażanie robotów konwersacyjnych i innych aplikacji opartych na modelach LLM wymaga przemyślanego projektu i monitorowania, by wychwycić takie ryzyka jak tendencyjność lub nieodpowiednia zawartość. Na przykład chatbot Tay firmy Microsoft został wycofany krótko po tym, jak ukazał się w 2016 roku, ponieważ publikował obraźliwe wpisy w serwisie Twitter (obecnie X), co przekładało się na toksyczne interakcje.

Jeśli chodzi o rozumowanie, to model LLM może poprawnie podać gęstość owocu i gęstość wody, gdy zapytasz o to oddzielnie. Jednak będzie miał problem z połączeniem tych faktów w celu określenia, czy owoc zatonie, czy będzie unosił się na wodzie. Model nie potrafi powiązać informacji, które posiada.

Zobaczmy, jak możemy stawić czoła tym wyzwaniom.

Jak można łagodzić ograniczenia modeli LLM?

Aby złagodzić negatywny wpływ ograniczeń modeli LLM, musimy zastosować techniki takie jak:

- **Pobieranie rozszerzonej generacji** (ang. *retrieval augmentation*). Ta technika umożliwia modelowi dostęp do różnych baz wiedzy w celu uzupełnienia nieaktualnych danych treningowych. Dzięki temu model ma zapewniony zewnętrzny kontekst, a ryzyko halucynacji jest mniejsze.
- **Łączenie w łańcuchy**. Ta technika pozwala układać w sekwencje takie zadania jak wyszukiwanie czy obliczenia.
- **Inżynieria monitów**. Ta technika polega na pieczołowitym tworzeniu monitów z podaniem niezbędnego kontekstu, co prowadzi do poprawnych odpowiedzi modelu.
- **Monitorowanie, filtrowanie i przegląd**. Ta technika wymaga ciągłego przeglądania pojawiających się problemów z wejściem i wyjściem aplikacji, by skutecznie je wykrywać. Następnie ręczne przeglądanie oraz filtry automatyczne poprawiają potencjalne problemy z danymi wyjściowymi. Ta technika wymaga następujących elementów:
  - **Filtry**, na przykład listy bloków, klasyfikatory wrażliwości oraz filtry zakazanych słów, mogą automatycznie zgłaszać błąd.
  - **Fundamentalne reguły** monitorują i filtrują nieetyczne i nieodpowiednie treści.
  - **Przeglądy przeprowadzane przez człowieka** zapewniają wgląd w działanie modelu i zwracane przez niego dane wyjściowe.
- **Pamięć**. Ta technika polega na zachowaniu kontekstu rozmowy przez zapisywanie danych z rozmów i kontekstu.
- **Dostrajanie**. Trening i dostrajanie modelu LLM na odpowiedniejszych danych dla określonej dziedziny i reguł. Dzięki temu można przystosować działanie modelu pod kątem określonych zadań.

Jeszcze raz warto podkreślić to, co już było powiedziane wcześniej — skalowanie surowego modelu nie poprawi rozumowania i nie doda innych brakujących możliwości. W tym celu należy zastosować osobne techniki takie jak inżynieria monitów, tok myślowy, które są niezbędne w celu pokonania luki kompozycyjnej. Podejścia, na przykład autosugestia, łagodzą ograniczenia przez zachęcanie modeli do metodycznego rozkładania problemów na mniejsze części.

Wykorzystanie takich narzędzi podczas treningu zapewnia modelowi umiejętności, których by nie nabył w wyniku samego szkolenia na danych. Monity dostarczają kontekst, łączenie w łańcuchy zapewnia wnioskowanie, a pobieranie rozszerzonej generacji dostarcza nowych faktów. Przez połączenie tych technik możemy zmienić stochastyczną papugę w rozumujący silnik.

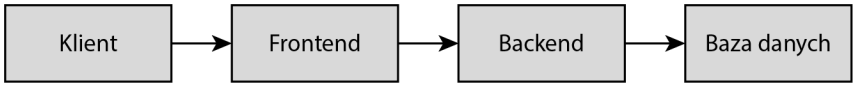
Rzetelna inżynieria monitów i dostrajanie przygotowuje modele do rzeczywistych przypadków użycia. Natomiast późniejsze ciągłe monitorowanie umożliwia wyłapanie pojawiających się problemów, zarówno poprzez automatyzacje, jak i przeglądy wykonywane przez człowieka. Filtry stoją na pierwszej linii obrony. Przyjęcie podstawowych reguł sztucznej inteligencji wspiera tworzenie kompetentnych modeli, które zachowują się w etyczny sposób. To kompleksowe podejście łączy w sobie przygotowanie, czujność i z natury korzystny projekt.

Połączenie modelu LLM z zewnętrznym źródłem danych zmniejsza ryzyko halucynacji i poprawia dokładność odpowiedzi opartych na najnowszych danych. Jednak bezpieczna integracja takich źródeł danych jak bazy danych przekłada się na bardziej złożony model. Frameworki takie jak LangChain ułatwiają to zadanie, jednocześnie zapewniając strukturę i nadzór nad odpowiedzialnym wykorzystaniem modeli LLM. Co więcej, tego typu narzędzia umożliwiają tworzenie modelowych monitów i źródeł danych, by pozbyć się deficytów oddzielnych modeli LLM. Poprzez staranne rozszerzanie modeli możemy tworzyć systemy sztucznej inteligencji, które wcześniej nie były zdolne do wykonywania wielu zadań z powodu wrodzonych ograniczeń. To prowadzi nas do kolejnego punktu naszej dyskusji.

Czym jest aplikacja oparta na modelu językowym?

Jeśli za pomocą specjalnego zestawu narzędzi połączymy model LLM z innymi narzędziami, nasze aplikacje napędzane dużymi modelami językowymi mogą zmieniać nasz cyfrowy świat. Często stosuje się technikę łączenia w łańcuchy pojedynczych lub kilku monitów wysyłanych do modeli LLM, ale można też korzystać z zewnętrznych serwisów (na przykład z API lub źródeł danych).

Tradycyjne oprogramowanie zwykle opiera się na wielowarstwowej architekturze, widocznej na rysunku 2.5.



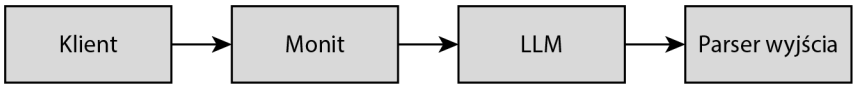
Rysunek 2.5. Tradycyjna aplikacja

Warstwa klienta obsługuje interakcję użytkownika z aplikacją. Warstwa frontendu obsługuje prezentację graficzną i logikę biznesową. Warstwa backendu przetwarza logikę, obsługuje API, wykonuje obliczenia itd. Na końcu warstwa bazodanowa zapisuje i przechowuje dane.

Natomiast **aplikacja oparta na modelu LLM** używa bardzo dużych modeli językowych, by rozumieć zapytania w języku naturalnym i generować odpowiedzi. Aplikacje oparte na modelu LLM zazwyczaj składają się z wymienionych elementów, którymi są:

- warstwa klienta w celu zbierania danych wejściowych od użytkownika — monitów lub decyzji;
- warstwa inżynierii monitów w celu tworzenia monitów będących przewodnikiem dla modelu;
- backend LLM do analizy monitów i wytwarzania właściwych odpowiedzi;
- warstwa analizująca dane wyjściowe (parser) w celu interpretacji odpowiedzi dostarczonych przez model LLM na potrzeby interfejsu aplikacji;
- dodatkowa, nieobowiązkowa integracja z zewnętrznymi usługami przez API, bazy danych i algorytmy rozumowania w celu poszerzenia możliwości modelu LLM.

W najprostszych przypadkach, frontendu, analiza tekstu i baza wiedzy nie są jawnie definiowane. Takie aplikacje składają się tylko z warstwy klienta, monitów i modelu LLM, tak jak pokazano na rysunku 2.6.



Rysunek 2.6. Prosta aplikacja oparta na modelu LLM

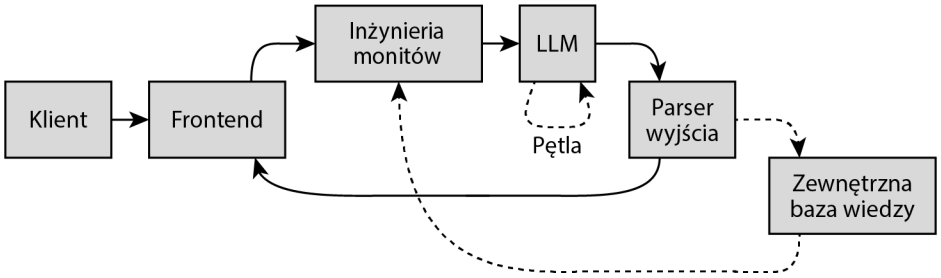
Aplikacje oparte na modelu LLM mogą być zintegrowane z zewnętrznymi usługami poprzez:

- API w celu uzyskania dostępu do narzędzi sieciowych i baz danych;
- zaawansowane algorytmy rozumowania na potrzeby złożonych łańcuchów logicznych;
- pobieranie rozszerzonej generacji z wykorzystaniem bazy wiedzy.

Pobieranie rozszerzonej generacji (RAG, ang. *retrieval augmented generation*), które omówimy szerzej w rozdziale 5. „Tworzenie robota konwersacyjnego na kształt ChatGPT", wzbogaca model LLM o wiedzę z zewnątrz. Te rozszerzenia zwiększają możliwości aplikacji opartych na modelach LLM, gdyż te nie muszą się już ograniczać do tego, co same wiedzą. Oto przykłady takich rozszerzeń:

- Wywoływanie funkcji umożliwia wysyłanie do API żądań z parametrami.
- Funkcje SQL umożliwiają konwersacyjne zapytania do bazy danych.
- Wykorzystanie algorytmów rozumowania, takich jak tok myślowy, umożliwia wielostopniową logikę.

Rysunek 2.7 przedstawia rozbudowaną aplikację opartą na modelu LLM.



Rysunek 2.7. Zaawansowana aplikacja oparta na modelu LLM

Jak widać na rysunku 2.7, warstwa klienta zbiera zapytania tekstowe i decyzje od użytkownika. Zadaniem inżynierii monitów jest stworzenie przewodnika dla modelu LLM z uwzględnieniem zewnętrznej wiedzy lub możliwości (lub wcześniejszych interakcji) bez zmieniania samego modelu. Backend LLM dynamicznie rozumie pytania i odpowiada na nie, zgodnie z tym jak został wytrenowany. Parser danych wyjściowych interpretuje tekst zwrócony przez model LLM na potrzeby frontendu. Baza wiedzy może wzbogacić informacje dostarczone przez model LLM, a także, podobnie jak w przypadku tradycyjnego backendu bazodanowego, dodać informacje.

Aplikacje oparte na modelu LLM są ważne z kilku względów.

- Backend LLM obsługuje język ludzki, pełen niuansów, bez sztywno zdefiniowanych reguł.
- Odpowiedzi mogą być spersonalizowane i umieszczone w kontekście na podstawie wcześniejszych interakcji.
- Zaawansowane algorytmy rozumowania umożliwiają złożone, wielostopniowe łańcuchy wnioskowania.
- Można otrzymać dynamiczne odpowiedzi oparte na informacji zwróconej przez model LLM lub na najnowszych informacjach uzyskanych w czasie rzeczywistym.

Kluczową zdolnością aplikacji opartych na modelach LLM jest rozumienie monitów wyrażonych w języku pełnym niuansów oraz zwracanie spójnych odpowiedzi, takich jakich udzieliłby człowiek. Dzięki temu możemy uzyskać bardziej naturalną interakcję użytkowników z aplikacją niż w przypadku tradycyjnego kodu.

Model LLM cechuje się zdolnościami językowymi podobnymi do człowieka, bez konieczności ręcznego pisania kodu. Innymi słowy, nie ma potrzeby przewidywania i pisania kodu dla każdego możliwego scenariusza. Integracja modeli LLM z zewnętrznymi serwisami, bazami wiedzy i algorytmami rozumowania ułatwia rozwój innowacyjnych aplikacji.

Jednak niezbędne są odpowiedzialne praktyki pracy z danymi. Dane osobowe nie powinny być przechowywane na publicznych platformach, a modele powinny być dostrajane, jeśli zajdzie taka potrzeba, wewnętrznie w danej organizacji. Zarówno frontend, jak i parser wyjścia mogą zawierać reguły moderowania i wzmacniania dotyczące działania aplikacji, prywatności i bezpieczeństwa. Przyszłe badania naukowe muszą zmierzyć się z wątpliwościami związanymi z potencjalnym złośliwym wykorzystaniem, tendencyjnością i ograniczeniami takich aplikacji.

W tej książce znajdziesz wiele przykładów aplikacji opartych na modelach LLM. Oto kilka z nich:

- **Chatboty i wirtualne asystenty**. Te aplikacje korzystają z modeli LLM takich jak ChatGPT w celu prowadzenia naturalnych rozmów z użytkownikami i pomagania w takich zadaniach jak planowanie grafiku, obsługa klienta i wyszukiwanie informacji.
- **Silniki inteligentnych wyszukiwarek**. Aplikacje oparte na modelach LLM mogą analizować wpisane w wyszukiwarce frazy wyrażone w języku naturalnym i zwracać właściwe wyniki.
- **Automatyczne tworzenie treści**. Aplikacje mogą wykorzystywać modele LLM na przykład w celu generowania treści artykułów, e-maili, kodu i innych treści w oparciu o monit tekstowy.
- **Odpowiadanie na pytania**. Użytkownicy mogą zapytać aplikację opartą na modelu LLM i szybko uzyskać wyczerpującą odpowiedź na bazie wiedzy modelu.
- **Analiza wydźwięku tekstu**. Za pomocą aplikacji LLM możesz analizować otrzymane od klientów informacje zwrotne, opinie i posty w serwisach społecznościowych. Taka aplikacja będzie w stanie podsumować odczucia autora tekstu i wyciągnąć główne kwestie poruszane w tekście.
- **Streszczenie tekstu**. Za pomocą backendu LLM możesz automatycznie streszczać długie dokumenty tekstowe i artykuły.
- **Analiza danych**. Modele językowe możesz wykorzystać do automatycznej analizy danych i wizualizacji w celu wyciągnięcia wniosków.
- **Generowanie kodu**. Możesz uzbroić programistów w asystenty LLM, by pomóc im rozwiązywać problemy podczas tworzenia oprogramowania.

Prawdziwa siła dużych modeli językowych nie tkwi w odizolowanych modelach LLM, ale w tych modelach LLM, które są połączone z innymi źródłami wiedzy i mocą obliczeniową. Framework LangChain ma zadanie umożliwiać tego typu integracje, ułatwiając w ten sposób tworzenie aplikacji osadzonych w kontekście i opartych na rozumowaniu. LangChain jest lekiem na bolączki związane z modelami LLM i zapewnia intuicyjny framework do tworzenia niestandardowych rozwiązań z wykorzystaniem przetwarzania języka naturalnego.

Co to jest LangChain?

LangChain, stworzony przez Harrisona Chase'a w 2022 roku, to framework z otwartym źródłem (ang. *open-source*) dla języka Python przeznaczony do budowy aplikacji opartych na modelach LLM. LangChain zapewnia programistom modułowe, łatwe w użyciu komponenty w celu łączenia modeli językowych z zewnętrznymi źródłami danych i usługami. Ten projekt otrzymał milionowe dofinansowania od takich organizacji jak Sequoia Capital i Benchmark, które finansowały również Apple, Cisco, Google, WeWork, Dropbox i inne odnoszące sukcesy firmy.

LangChain zapewnia komponenty wielokrotnego użytku i wstępnie ułożone łańcuchy, co znacznie ułatwia tworzenie zaawansowanych aplikacji opartych na modelach LLM. Modułowa architektura tego frameworku zapewnia dostęp do modeli LLM i zewnętrznych usług w ujednoliconym interfejsie. Programiści mogą łączyć te elementy do obsługi złożonych przepływów pracy.

Budowa wpływowych aplikacji opartych na modelach LLM wiąże się z wyzwaniami, między innymi z inżynierią monitów, zmniejszaniem tendencyjności, zmianą prototypu w wersję produkcyjną czy integracją danych zewnętrznych. Budowy aplikacji z LangChain można się stosunkowo szybko nauczyć, dzięki uproszczeniom (abstrakcjom) i strukturze modułowej.

LangChain umożliwia nie tylko korzystanie z API modelu LLM, ale poprzez agenty i pamięć ułatwia również zaawansowane interakcje, takie jak kontekst konwersacji oraz ciągłość rozmowy. W ten sposób chatboty mogą między innymi zbierać dane z zewnątrz.

To w szczególności obsługa łańcuchów, agenty, narzędzia i pamięć frameworku LangChain umożliwiają programistom tworzenie aplikacji, które mogą wchodzić w interakcje ze środowiskiem w bardziej zaawansowany sposób oraz przechowywać i ponownie używać informacji. Dzięki swojej modułowości LangChain sprawia, że budowanie złożonych aplikacji, które można przystosować do wielu dziedzin, jest łatwe. Wsparcie planów działania i strategii zwiększa wydajność i stabilność aplikacji. Obsługa pamięci i dostęp do zewnętrznych informacji zmniejszają ryzyko halucynacji, przez co aplikacja może cieszyć się większym zaufaniem.

Główne zalety frameworku LangChain dla programistów to:

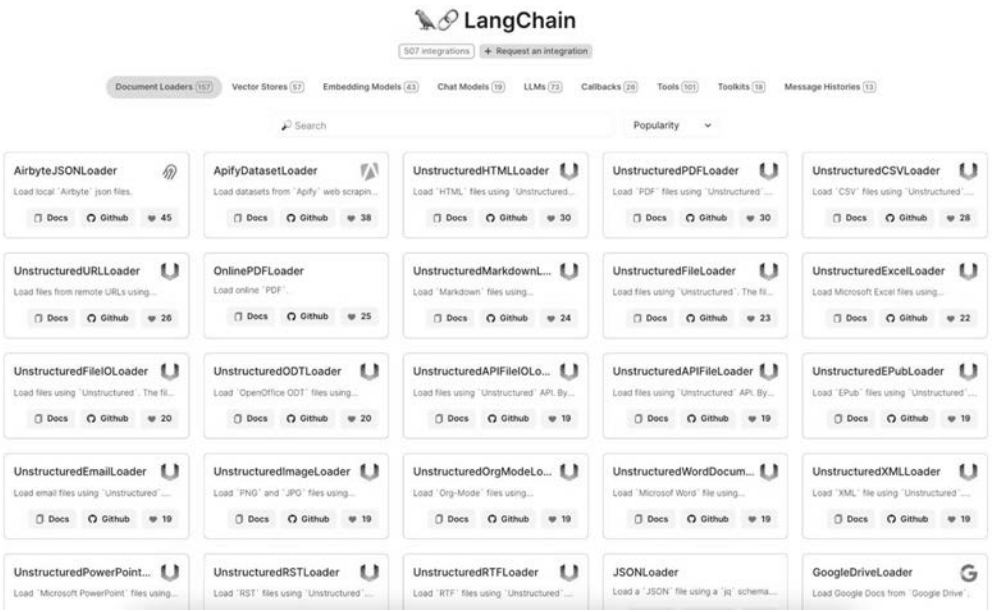
- **architektura modułowa** dla elastycznych i łatwych do przystosowania integracji modeli LLM;
- **łączenie w łańcuchy** wielu usług, dzięki czemu programiści nie są ograniczeni tylko do samych modeli LLM;
- interakcje z agentem nastwione na cel zamiast osobnych żądań i wywołań;
- **pamięć i trwałość** dla zapewnienia stanowości podczas działania;
- **dostęp na licencji open source** i wsparcie społeczności.

Jak już wspomniano wcześniej, LangChain ma otwarte źródło i jest napisany w języku Python, choć istnieją projekty towarzyszące: *LangChain.js* zaimplementowany w języku JavaScript, a dokładniej — w TypeScript, oraz nowo powstały *Langchain.rb* dla języka Ruby z interpreterem do uruchamiania kodu. W tej książce skupię się na frameworku dla Pythona.

Choć dokumentacja, kursy i pomoc społeczności przyspieszają naukę tworzenia aplikacji, to do tego, aby stać się ekspertem w stosowaniu modeli LLM, potrzeba czasu i wysiłku. Wielu programistów może zniechęcić proces nauki, a wskutek tego nie będą w stanie w pełni wykorzystywać modeli LLM.

Na platformie Discord znajdziesz wiele ożywionych dyskusji, blogów i regularnych spotkań odbywających się w wielu miastach na świecie. Natomiast na stronie z dokumentacją LangChain znajdziesz chatbota, ChatLangChain, który odpowie Ci na pytania związane z dokumentacją frameworku LangChain. Ten robot konwersacyjny powstał przy użyciu LangChain właśnie i FastAPI!

LangChain ma wiele rozszerzeń i duży ekosystem, który cały czas się rozrasta. Jak już wspomniano wcześniej, istnieje już ogromna liczba integracji LangChain, a każdego tygodnia są dodawane nowe. Na rysunku 2.8 pokazano kilka integracji (źródło: *https://python.langchain.com/docs/integrations/platforms/*).



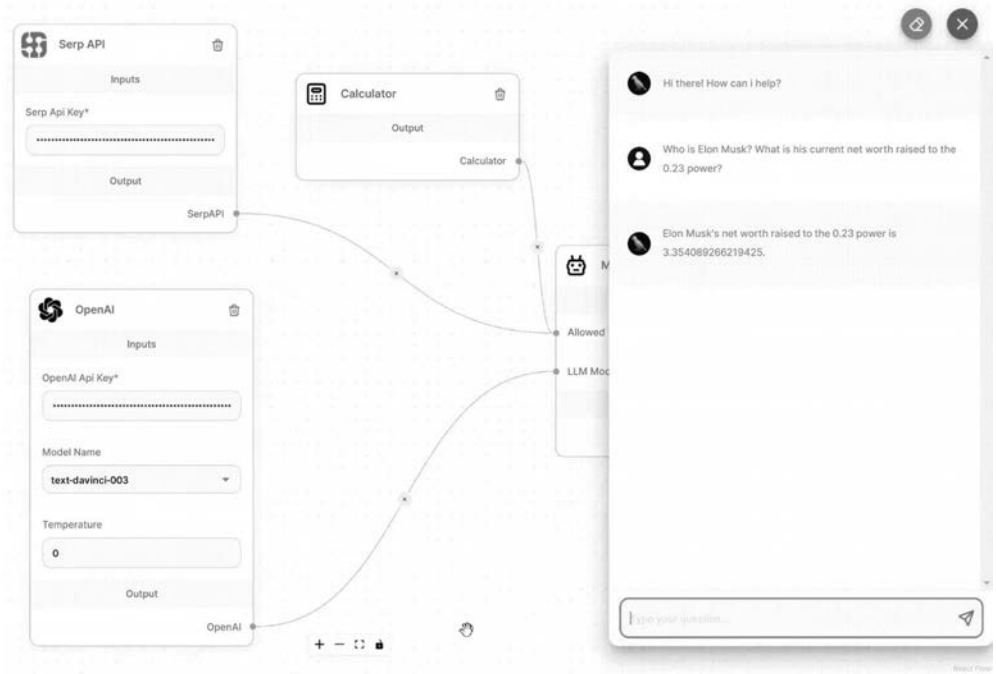
Rysunek 2.8. Integracje LangChain (stan na wrzesień 2023)

Platforma LangSmith, będąca częścią ekosystemu LangChain, zapewnia debugowanie, testowanie i monitorowanie dla aplikacji opartych na modelach LLM. Na przykład programiści mogą szybko debugować nowe łańcuchy, w czym pomaga im szczegółowy widok ścieżek wykonywania się programu. Co więcej, można testować alternatywne monity oraz modele LLM z różnymi zestawami danych, by zagwarantować jakość i spójność. Z kolei analiza użytkowania pomaga w podejmowaniu opartej na danych decyzji dotyczących optymalizacji.

LlamaHub i LangChainHub dostarczają darmowe biblioteki z elementami wielokrotnego użycia do uproszczonej budowy zaawansowanych systemów LLM. **LlamaHub** to biblioteka z funkcjami wczytującymi dane, stworzonymi przez społeczność LlamaIndex. Ta biblioteka zapewnia zestaw narzędzi do łatwego łączenia modeli LLM z zewnętrznymi źródłami danych. LlamaHub upraszcza tworzenie niestandardowych agentów w celu odblokowania możliwości modeli LLM.

LangChainHub to centralne repozytorium z udostępnionymi artefaktami. Znajdziesz tam między innymi monity, łańcuchy i agenty dla LangChain. Podobnie jak Hugging Face Hub, LangChainHub to jedno źródło z wysokiej jakości elementami składowymi złożonych aplikacji LLM. Na razie LangChainHub ogranicza się do zestawu monitów, które można zapożyczać do swoich projektów. W przyszłości planowane jest dodanie wsparcia dla łańcuchów, agentów i innych kluczowych elementów frameworku LangChain.

LangFlow i **Flowise** to interfejsy użytkownika umożliwiające łączenie w łańcuch komponentów LangChain. Wystarczy przeciągnąć na obszar roboczy komponenty wybrane z paska bocznego i je połączyć, a w ten sposób powstanie diagram sekwencji działań. Można dzięki temu łatwo eksperymentować i tworzyć prototypy, co widać na rysunku 2.9 przedstawiającym zrzut ekranu z Flowise (źródło: *https://github.com/FlowiseAI/Flowise*).



Rysunek 2.9. Interfejs użytkownika Flowise z agentem używającym modelu LLM, kalkulatorem i narzędziem do wyszukiwania

Na rysunku 2.9 widać agenta (agenty zostaną omówieni później w tym rozdziale) połączonego z interfejsem wyszukiwania (**Serp API**), modelem LLM i kalkulatorem. LangChain i LangFlow mogą być wdrażane lokalnie, na przykład za pomocą biblioteki Chainlit, lub na innych platformach, na przykład na Google Cloud. Biblioteka `langchainserve` pomaga wdrażać zarówno LangChain, jak i LangFlow w postaci aplikacji LLM jako usługi w chmurze Jina AI za pomocą jednego polecenia.

Choć LangChain jest stosunkowo nowy, otwiera możliwości zaawansowanych zastosowań aplikacji opartych na modelach LLM, gdyż można je połączyć z takimi komponentami jak pamięć, tworzenie łańcuchów czy agenty. LangChain ma na celu ułatwienie wszystkich skomplikowanych etapów tworzenia aplikacji LLM. Dlatego teraz skupimy się na sposobie działania LangChain i na jego komponentach.

Kluczowe komponenty LangChain

Łańcuchy, agenty, pamięć i narzędzia umożliwiają tworzenie zaawansowanych aplikacji LLM, które wykraczają poza zwykłe wysyłanie żądań do API modelu LLM. W kolejnych punktach omówimy te kluczowe koncepcje, zastanowimy się, jak umożliwiają tworzenie sprawnych systemów przez połączenie modeli językowych z zewnętrznymi danymi i usługami.

W tym rozdziale nie będziemy się zagłębiać we wzorce implementacji, jednak omówimy bardziej szczegółowo, do czego służą poszczególne komponenty. Po przeczytaniu tego rozdziału powinieneś rozumieć, jak budować systemy z LangChain. Zacznijmy od łańcuchów!

Czym są łańcuchy?

Łańcuchy są kluczową koncepcją frameworku LangChain i służą do tworzenia potoków modułowych komponentów wielokrotnego użytku. I tak na przykład programiści mogą łączyć kilka monitów wysyłanych do modelu LLM z innymi komponentami, by tworzyć złożone aplikacje na przykład na potrzeby chatbota, wyciągać i analizować dane. Ogólnie mówiąc, łańcuch to sekwencja wywołań komponentów, które mogą zawierać inne łańcuchy. Najbardziej podstawowym przykładem łańcucha jest najprawdopodobniej `PromptTemplate`, który przekazuje sformatowaną odpowiedź do modelu językowego.

Łączenie monitów w łańcuchy (ang. *prompt chaining*) to technika, której można używać do poprawy wydajności aplikacji tworzonych w LangChain. Polega na łączeniu w łańcuchy wielu monitów w celu automatycznego uzupełniania bardziej złożonych odpowiedzi. Bardziej skomplikowane łańcuchy łączą w sobie takie narzędzia jak `LLMMath`, dla zapytań matematycznych, czy `SQLDatabaseChain`, do wysyłania zapytań do bazy danych. Takie łańcuchy nazywamy łańcuchami użytkowymi (ang. *utility chains*), ponieważ łączą z sobą modele językowe i konkretne narzędzia.

Łańcuchy mogą nawet przestrzegać ustalonych zasad, na przykład mogą łagodzić toksyczne odpowiedzi lub je normować zgodnie z regułami etyki. LangChain implementuje łańcuchy, by upewnić się, że zwracane dane nie są szkodliwe i nie łamią zasad umiarkowanych odpowiedzi wyznaczonych przez OpenAI (`OpenAIModerationChain`) lub nie są nieetyczne, niezgodne z prawem lub innymi, niestandardowymi regułami (`ConstitutionalChain`).

Łańcuch `LLMCheckerChain` za pomocą techniki autorefleksji sprawdza zwracane wyrażenia. `LLMCheckerChain` może zapobiegać halucynacjom i zmniejszyć liczbę niepoprawnych odpowiedzi poprzez sprawdzanie założeń stojących za dostarczonymi wyrażeniami i pytaniami. W artykule napisanym przez naukowców z uniwersytetu Carnegie Mellon, Allen Institute, Uniwersytetu Waszyngtońskiego, firm NVIDIA, UC SanDiego i Google Research w maju 2023 (*SELF-REFINE: Iterative Refi nement with Self-Feedback*) można znaleźć stwierdzenie, że ta strategia może zwiększyć średnio o 20% wydajność wykonywanych zadań, w tym generowanie kwestii dialogów, rozumowanie matematyczne i rozumienie kodu.

Już zaledwie kilka łańcuchów może podejmować autonomiczne decyzje. Podobnie do agentów, łańcuchy-routery mogą decydować o doborze narzędzi na podstawie ich opisów. Łańcuch `RouterChain` może dynamicznie wybierać sposób uzyskania informacji, na przykład za pomocą monitów lub indeksów.

Łańcuchy mają wiele ważnych zalet:

- **Modułowość**. Logika jest podzielona na komponenty wielokrotnego użytku.
- **Kompozycyjność**. Komponenty mogą być dowolnie łączone w łańcuchy.
- **Czytelność**. Wyraźnie widać każdy krok w potoku.
- **Łatwość utrzymywania**. Kroki mogą być dodawane, usuwane i zamieniane miejscami.
- **Możliwość wielokrotnego wykorzystania**. Często używane łańcuchy staja się łańcuchami, które można konfigurować.
- **Integracja narzędzi**. Do swojego projektu można łatwo dołączać modele LLM, bazy danych, API itd.
- **Produktywność**. Można szybko tworzyć prototypy konfigurowalnych łańcuchów.

To wszystko razem umożliwia enkapsulację złożonych przepływów pracy w łatwe do zrozumienia i stosowania potoki ułożone w łańcuch.

Zwykle tworzenie łańcucha LangChain wymaga rozbicia przepływu pracy na pojedyncze kroki logiczne, takie jak wczytywanie danych, przetwarzanie, wysyłanie monitów do modelu. Dobrze zaprojektowane łańcuchy zawierają komponenty wykonujące jeden rodzaj zadań. Kroki powinny być bezstanowymi funkcjami w celu zwiększenia możliwości wielokrotnego zastosowania. Konfiguracja łańcuchów powinna jak najbardziej umożliwiać dostosowanie łańcucha do różnych potrzeb. Niezawodne łańcuchy cechują się solidną obsługą błędów i wyjątków. Monitorowanie i zapisywanie akcji w rejestrze (ang. *logging*) może być obsługiwane przez różne mechanizmy, w tym przez wywołania zwrotne.

Teraz opowiemy więcej o agentach i o tym, jak podejmują decyzje!

Czym są agenty?

Agenty to kluczowa koncepcja LangChain służąca do tworzenia systemów dynamicznie współpracujących z użytkownikami i środowiskami. Agent jest niezależnym programem, który może podejmować działania, aby wykonać postawione przed nim zadanie lub osiągnąć zadany cel.

Łańcuchy i agenty to podobne koncepcje, dlatego warto podać różnice między nimi. LangChain cechuje się przede wszystkim tym, że łączy modele LLM z innymi komponentami. Robią to zarówno łańcuchy, jak i agenty, ale w inny sposób. Oba rozszerzają modele LLM, ale agenty zarządzają łańcuchami, podczas gdy łańcuchy ustawiają w szeregu moduły niższego poziomu. Łańcuchy definiują logikę wielokrotnego użytku przez tworzenie sekwencji komponentów, natomiast agenty wykorzystują łańcuchy, by osiągnąć dany cel. Agenty łączą łańcuchy i zarządzają nimi. Agenty obserwują środowisko i na podstawie obserwacji decydują, który łańcuch wykonać, uruchamiają go i powtarzają.

Agenty decydują, jakie podjąć działania z wykorzystaniem modeli LLM jako silników rozumowania. Do modelu LLM są wysyłane monity za pomocą dostępnych narzędzi, danych wprowadzonych przez użytkownika i poprzednich kroków. Następnie wybierane jest kolejne zadanie lub końcowa odpowiedź.

Narzędzia (omówione później w tym rozdziale) to funkcje wywoływane przez agenta w celu podjęcia rzeczywistych zadań. Aby agent mógł skutecznie osiągnąć swój cel, musimy mu dostarczyć odpowiednich narzędzi oraz odpowiednio je opisać.

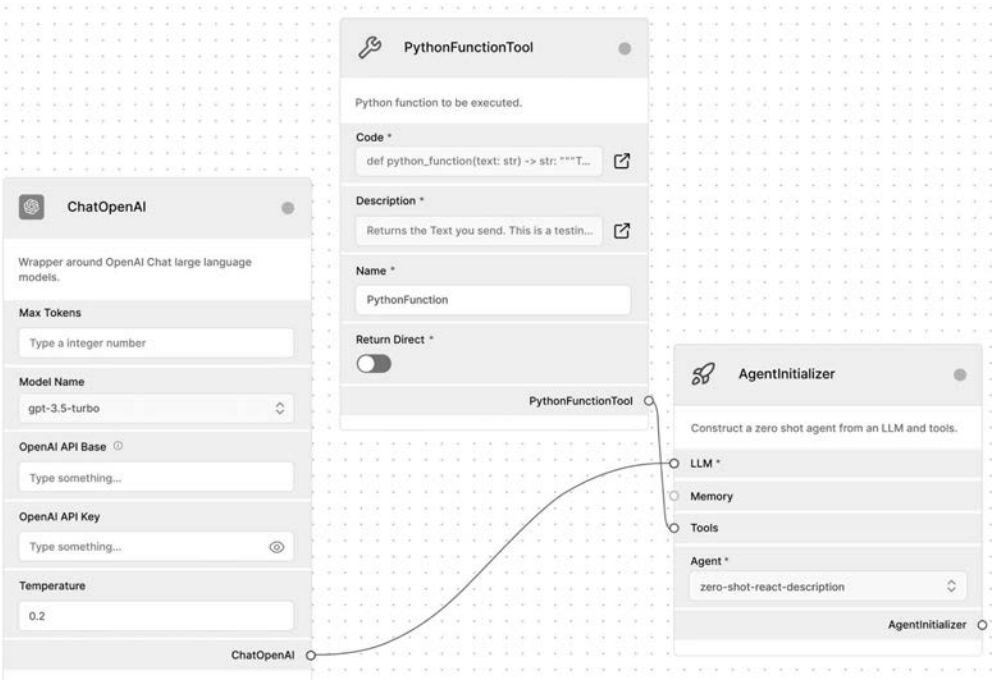
Agent działający cały czas zarządza pętlą wysyłającą monity do agentów, uruchamiającą narzędzia i zwracającą obserwacje. W ten sposób są wykonywane zadania o mniejszym stopniu złożoności, między innymi obsługa błędów, zapis z działania programu w rejestrze i parsowanie.

Agenty mają kilka kluczowych zalet:

- **Działania zorientowane na cel**. Agenty mogą planować łańcuchy logiczne na potrzeby określonego celu.
- **Dynamiczne odpowiedzi**. Obserwowanie zmian środowiska pozwala agentom reagować i się przystosowywać.
- **Stanowość**. Agenty mogą zapisywać w pamięci informacje i kontekst interakcji.
- **Solidność**. Błędy można obsługiwać przez wychwytywanie wyjątków i uruchamianie alternatywnych łańcuchów.
- **Budowa**. Logika agenta łączy w sobie łańcuchy komponentów wielokrotnego użytku.

To wszystko umożliwia agentom obsługę złożonych z wielu kroków przepływów pracy i aplikacji, które są w niemal ciągłej interakcji z użytkownikiem, takich jak chatboty.

W punkcie traktującym o ograniczeniach modeli LLM widzieliśmy, że w obliczeniach matematycznych prosty kalkulator radzi sobie lepiej niż model z miliardami parametrów. W takich przypadkach agent może zdecydować, czy przekazać to zadanie kalkulatorowi, czy interpreterowi Pythona. Na rysunku 2.10 widać prostą aplikację, w której agent połączony jest zarówno z modelem OpenAI, jak i funkcją Pythona.



Rysunek 2.10. Prosta aplikacja LLM z funkcją napisaną w Pythonie zwizualizowana w LangFlow

W zależności od danych wejściowych agent może zdecydować o wywołaniu funkcji Pythona. Każdy agent decyduje, jakiego narzędzia użyć i kiedy. W rozdziale 4. „Budowanie skutecznych agentów" zobaczymy dokładniej, jak to działa.

Kluczowym ograniczeniem agentów i łańcuchów jest ich bezstanowość — każde działanie agenta jest odosobnione, przez co nie jest on w stanie zachować poprzedniego kontekstu. Dlatego tutaj ważną rolę odgrywa pamięć. Pamięć w LangChain to nic innego jak przechowywanie informacji z działania łańcuchów, by zapewnić stanowość.

Czym jest pamięć?

W LangChain pamięć odnosi się do przechowywania stanu między kolejnymi działaniami łańcucha lub agenta. Jeśli rzetelnie podejdziemy do kwestii pamięci, to programistom będzie łatwiej budować interaktywne i konwersacyjne aplikacje. Na przykład przechowywanie historii rozmów na czacie pozytywnie wpływa na spójność i poprawność odpowiedzi.

Zamiast traktować każde dane wyjściowe od użytkownika jako osobny monit, łańcuchy mogą przekazywać dane z rozmowy do modeli, by zapewnić spójność. Co więcej, agenty mogą zapisywać w pamięci fakty o świecie, relacje i wnioski. Ta wiedza jest cały czas dostępna, nawet jeśli rzeczywiste warunki uległy zmianie, dzięki czemu agent zna kontekst. Cele i ukończone zadania zapisane w pamięci pozwalają agentom śledzić postęp wieloetapowych zadań przez całą rozmowę. Ponadto zapisywanie informacji w pamięci zmniejsza liczbę wysyłanych do modelu LLM próśb o te same informacje. Dzięki temu użycie API jest mniejsze, a co za tym idzie, koszty są mniejsze, a agent czy łańcuch wciąż znają kontekst.

LangChain zapewnia standardowy interfejs dla pamięci, integrację z różnymi miejscami do przechowywania danych, między innymi bazami danych oraz wzorcami projektowymi dla skutecznego wykorzystania pamięci w łańcuchach i agentach.

Mamy do wyboru kilka możliwości, na przykład:

- `ConversationBufferMemory` zapisuje wszystkie wiadomości w historii modelu, ale wydłuża opóźnienia w działaniu aplikacji i zwiększa koszt.
- `ConversationBufferWindowMemory` przechowuje tylko ostatnie wiadomości.
- `ConversationKGMemory` streszcza rozmowy w postaci grafów wiedzy w celu integracji z monitami.
- `EntityMemory` zapisuje informacje w bazie danych, zachowując fakty i stan agenta.

Poza tym LangChain integruje wiele różnych baz danych dla trwałego przechowywania danych.

- Relacyjne bazy danych takie jak PostgreSQL czy SQLLite umożliwiają relacyjne modelowanie danych.
- Nierelacyjne bazy danych takie jak MongoDB i Cassandra ułatwiają pracę ze skalowalnymi nieustrukturyzowanymi danymi.
- Redis zapewnia lokalną bazę danych dla pamięci podręcznej o wysokiej wydajności.
- Usługi chmurowe takie jak AWS DynamoDB ściągają z nas obowiązek przygotowywania infrastruktury.

Poza bazami danych są jeszcze serwery przygotowane specjalnie pod kątem przechowywania danych, na przykład Remembrall i Motörhead, które oferują zoptymalizowany kontekst konwersacji. Dobór odpowiedniego podejścia do pamięci zależy od takich czynników jak czas przechowywania danych, relacje między danymi, skala i zasoby, jednak niezawodne zapamiętywanie stanu jest kluczowe dla interaktywnych aplikacji konwersacyjnych.

Integrowanie pamięci przy użyciu LangChain, od krótkiego przechowywania danych w pamięci podręcznej po bazy danych z długim czasem przechowywania danych, umożliwia budowanie znających kontekst agentów z opcją zapisywania stanu. Tworzenie skutecznych wzorców pamięci to droga ku nowej generacji wiele potrafiących i niezawodnych systemów sztucznej inteligencji. LangChain oferuje długą listę narzędzi, z których możemy korzystać we własnych aplikacjach. Nie jestem w stanie wymienić wszystkich w jednym krótkim punkcie, ale postaram się przynajmniej omówić kilka z nich.

Dostępne narzędzia

Narzędzia dostarczają modułowe interfejsy dla agentów, by integrować zewnętrzne usługi, na przykład bazy danych czy API. Zestawy narzędzi grupują narzędzia współdzielące zasoby. Narzędzie mogą być łączone z modelami w celu poszerzenia możliwości modeli. LangChain oferuje między innymi narzędzia do wczytywania dokumentów, indeksowania, przechowywania wektorów, co ułatwia pobieranie i przechowywanie danych dla modeli LLM i nie tylko.

Mamy do dyspozycji wiele narzędzi, a to tylko kilka z nich:

- **Tłumacz maszynowy**. Model językowy może korzystać z tłumacza językowego, aby lepiej zrozumieć i przetworzyć tekst w różnych językach. To narzędzie sprawia, że modele językowe nieprzeznaczone do tłumaczeń rozumieją pytania w różnych językach i odpowiadają na nie.
- **Kalkulator**. Model językowy może korzystać z prostego kalkulatora w celu rozwiązywania zadań matematycznych. Kalkulator obsługuje podstawowe działania arytmetyczne, dzięki czemu model może poprawnie rozwiązywać zadane obliczenia matematyczne w zestawach danych przeznaczonych do rozwiązywania problemów matematycznych.
- **Mapy**. Przez połączenie do Bing Map API lub podobnych usług modele językowe mogą pobierać informacje o lokalizacji, planować drogę, obliczać odległości i podawać szczegóły o najbliższych punktach na mapie, które mogą nas interesować.
- **Pogoda**. API pogodowe w czasie rzeczywistym dostarczają do modeli językowych informacje o pogodzie dla dowolnego miasta na świecie. Modele mogą odpowiadać na pytania o aktualne warunki pogodowe lub o prognozę dla podanych lokalizacji.
- **Giełda**. Po połączeniu z API giełdowymi, takimi jak Alpha Vantage, modele językowe mogą wysyłać żądania o określone dane giełdowe, na przykład o ceny otwarcia czy zamknięcia, o najwyższe i najniższe ceny i o wiele więcej.

- **Slajdy**. Modele językowe wyposażone w narzędzia do tworzenia slajdów są w stanie tworzyć prezentacje za pomocą API, na przykład biblioteki *python-pptx*, lub pobierać rysunki z internetu odpowiednie dla umieszczonych na slajdach treści. Te narzędzia ułatwiają wykonywanie zadań związanych z tworzeniem prezentacji, które wciąż są popularne i wymagane w wielu branżach.
- **Przetwarzanie tabel**. API stworzone za pomocą ramek danych (ang. *data frames*) z biblioteki *pandas* daje modelom możliwość przeprowadzania analizy danych i wizualizacji opartych na tabelach. Dzięki połączeniu z tymi narzędziami modele mogą zapewniać użytkownikom uproszczoną i naturalną pracę z tabelarycznymi danymi.
- **Grafy wiedzy**. Modele językowe mogą wysyłać zapytania do grafów wiedzy za pomocą API. I tak mogą na przykład znajdować odpowiednie jednostki lub relacje, wysyłać zapytania w SPARQL i pobierać wyniki. Te narzędzia pomagają odpowiadać na pytania na podstawie merytorycznej wiedzy przechowywanej w grafach wiedzy.
- **Silnik wyszukiwarki**. Dzięki API silników wyszukiwarek, na przykład Bing Search, modele językowe mogą z nimi współpracować, by wyciągać informacje i podawać odpowiedzi na pytania w czasie rzeczywistym. Te narzędzia wzbogacają model o możliwość zbierania informacji z sieci, dzięki czemu mogą podawać dokładniejsze odpowiedzi.
- **Wikipedia**. Modele językowe wyposażone w narzędzia wyszukujące informacje w Wikipedii mogą wyszukiwać określone hasła, wyszukiwać słowa kluczowe na stronie lub rozróżniać pojęcia o podobnej nazwie. Te narzędzia umożliwiają modelowi odpowiadanie na pytania z wykorzystaniem treści z Wikipedii.
- **Zakupy online**. Dzięki połączeniu modeli językowych z narzędziami obsługującymi zakupy przez internet modele mogą między innymi wyszukiwać produkty, wczytywać szczegółowe informacje o nich, wybierać cechy produktów, przeglądać strony z ofertami oraz decydować o zakupie na podstawie dostarczonych przez użytkownika instrukcji.

Poza tym mamy do dyspozycji inne narzędzia. AI Painting na przykład sprawia, że modele językowe mogą generować obrazy z wykorzystaniem generatywnych modeli sztucznej inteligencji generujących obraz. 3D Model Construction umożliwia modelom językowym tworzenie trójwymiarowych modeli przy użyciu zaawansowanych silników renderujących obiekty 3D. Chemical Properties z pomocą API takiego jak na przykład PubChem pomaga odpowiadać na pytania o właściwości chemiczne. Na koniec warto wspomnieć również o narzędziach do pracy z bazami danych, które to narzędzia umożliwiają wysyłanie do bazy zapytań wyrażonych za pomocą języka naturalnego zamiast wyrażeń SQL.

Te różne narzędzia zapewniają modelom językowym dodatkowe funkcjonalności i możliwości wykraczające poza przetwarzanie tekstu. Po połączeniu się z tymi narzędziami poprzez API modele językowe mają bogatsze zastosowania i mogą służyć między innymi do tłumaczeń, rozwiązywania zadań matematycznych, odpowiadania na pytania związane z danym położeniem, sprawdzania pogody, analizy giełdy, tworzenia slajdów, przetwarzania i analizowania danych tabelarycznych, generowania obrazów, przetwarzania tekstu na mowę, a także do innych, bardziej wyspecjalizowanych zadań.

Wszystkie te narzędzia mogą zapewnić zaawansowaną funkcjonalność sztucznej inteligencji, a dostęp do nich jest praktycznie nieograniczony. Możemy z łatwością tworzyć różne narzędzia poszerzające wachlarz zastosowań modeli językowych, a poza tym umożliwiają one modelom językowym skuteczne i wydajne wykonywanie różnych rzeczywistych zadań.

Po omówieniu łańcuchów, agentów, pamięci i narzędzi zbierzmy to wszystko w całość i zobaczmy, jak LangChain wykorzystuje je w działaniu.

Jak działa LangChain?

Framework LangChain ułatwia budowanie zaawansowanych aplikacji LLM, ponieważ zapewnia modułowe komponenty ułatwiające łączenie modeli językowych z innymi danymi i usługami. Ten framework rozdziela umiejętności na moduły obsługujące różne czynności, od podstawowej interakcji z modelem LLM po zaawansowane rozumowanie.

Możemy łączyć poszczególne komponenty w potoki, nazywane również łańcuchami, które układają w sekwencję następujące akcje:

- wczytywanie dokumentów,
- osadzanie na potrzeby uzyskania podobnych informacji,
- wysyłanie zapytań do modeli LLM,
- analizowanie danych wyjściowych,
- zapis w pamięci.

Łańcuchy dopasowują moduły do założeń aplikacji, natomiast agenty korzystają z łańcuchów, aby osiągnąć cel postawiony przez użytkownika w trakcie interakcji z aplikacją. Moduły ciągle wykonują zadania na podstawie obserwacji, planują optymalne łańcuchy logiczne i zarządzają pamięcią, gdzie zapisane są dane z konwersacji.

Oto kilka przykładowych modułów, od najprostszych do tych bardziej zaawansowanych:

- **Modele LLM i modele konwersacyjne** zapewniają interfejs dla modułów językowych takich jak GPT-3. Obsługują synchroniczne, napływające strumieniowo i grupowe żądania.
- **Moduły wczytujące dokumenty** pobierają dane ze źródeł i umieszczają je w dokumentach tekstowych z metadanymi. Dzięki temu można wczytywać pliki, strony internetowe, filmy itp.
- **Moduły przekształcające dane** poprzez podział, łączenie, filtrowanie, tłumaczenie itp. Pomagają przystosować dane dla modeli.
- **Osadzenia tekstu** tworzą reprezentację wektorową tekstu na potrzeby wyszukiwania znaczeniowego (ang. *semantic search*). Mamy różne metody dla osadzeń dokumentów i monitów.
- **Bazy wektorowe** przechowują wektory osadzonych dokumentów dla skuteczniejszego wyszukiwania i uzyskania podobnych danych.
- **Pozyskiwacze** (ang. *retrievers*) to interfejsy służące do zwracania dokumentów jako odpowiedzi na monit. Mogą wykorzystywać bazy wektorowe.

- **Narzędzia** to interfejsy używane przez agentów do współpracy z zewnętrznymi systemami.
- **Agenty** to systemy nastawione na cel, wykorzystujące modele LLM, aby planować kolejne działania na podstawie obserwacji własnego środowiska.
- **Zestawy narzędzi** to grupy narzędzi współdzielących takie zasoby jak bazy danych.
- **Pamięć** umożliwia zapisywanie i przechowywanie informacji z rozmów i przepływów pracy przez czytanie/zapisywanie danych z sesji.
- **Wywołania zwrotne** są doczepione do różnych kroków potoków, dzięki czemu możemy zapisywać przebieg działania programu w rejestrze, monitorować, wysyłać dane strumieniowo itp. Wywołania zwrotne umożliwiają monitorowanie łańcuchów.

Wszystkie wymienione wyżej możliwości ułatwiają budowanie w LangChain solidnych, wydajnych i wielofunkcyjnych aplikacji opartych na modelach LLM. Te moduły różnią się między sobą poziomem złożoności i istotności, więc warto powiedzieć o nich coś więcej.

LangChain oferuje interfejsy do łączenia się z modelami LLM (na przykład GPT-3) i wysyłania do nich monitów. Interfejsy obsługują asynchroniczne żądania, strumieniowe wysyłanie odpowiedzi i monity grupowe. Dzięki temu mamy do dyspozycji elastyczne API do integrowania różnych modeli językowych.

Choć framework LangChain nie zapewnia samych modeli, wspiera integracje za pomocą klas opakowujących (ang. *wrapper*) z modelami LLM od różnych dostawców. Obecnie LangChain obsługuje modele utworzone między innymi przez OpenAI, HuggingFace, Azure i Anthropic. Dzięki ustandaryzowanemu interfejsowi możemy bez wysiłku zamieniać modele, by zaoszczędzić pieniądze, energię i uzyskać lepszą wydajność. W rozdziale 3. „Rozpoczęcie pracy z LangChain" przyjrzymy się temu bliżej.

Podstawowym blokiem LangChain jest klasa monitu, która umożliwia interakcje użytkowników z modelami LLM odbywającą się przy użyciu zwięzłych instrukcji lub przykładów. Inżynieria monitów pomaga zoptymalizować monity pod kątem wydajności modelu. Szablony dają elastyczność, jeśli chodzi o dane wejściowe, a zestaw dostępnych monitów jest sprawdzany w boju przez wiele różnego rodzaju aplikacji. W rozdziale 3. „Rozpoczęcie pracy z LangChain" po raz pierwszy przyjrzymy się bliżej monitom, a inżynierią monitów zajmiemy się w rozdziale 8. „Niestandardowe modele LLM i zwracane przez nie rezultaty".

Moduły wczytujące dokumenty pozwalają przyjmować dane z różnych źródeł i zapisywać je w postaci dokumentów tekstowych z metadanymi. Możemy następnie dzielić, łączyć, filtrować, tłumaczyć itd. te dane przy użyciu modułu przetwarzającego dokumenty. Te narzędzia przystosowują zewnętrzne dane na potrzeby modeli LLM.

Moduły wczytujące dokumenty zawierają narzędzia do przechowywania danych, interakcji z zewnętrznymi systemami, na przykład wyszukiwarkami czy bazami danych, a co najważniejsze — do odzyskiwania danych. Wspomniane moduły obsługują między innymi tego typu dokumenty: dokumenty utworzone w programie Microsoft Word (*.docx*), dokumenty HTML (ang. *HyperText Markup Language*, hipertekstowy język znaczników) i inne popularne formaty jak pliki PDF, pliki tekstowe, pliki w formacie JSON i CSV. Inne narzędzia mogą wysyłać e-maile do potencjalnych klientów, publikować zabawne wpisy

dla osób obserwujących jakiś profil w mediach społecznościowych lub wysyłać wiadomości w serwisie Slack do Twoich współpracowników, ale o tym więcej w rozdziale 5. „Tworzenie robota konwersacyjnego na kształt ChatGPT".

Modele osadzające tekst tworzą wektorowe reprezentacje tekstu z uchwyceniem znaczenia. W rezultacie wyszukiwanie znaczeniowe jest możliwe i po wpisaniu danego tekstu możemy znaleźć najbardziej zbliżone reprezentacje wektorowe. Wektorowe bazy danych indeksują wektory z osadzonymi dokumentami, by szybko i skutecznie można było uzyskać podobne informacje.

Wektorowe bazy danych wkraczają do akcji, gdy zachodzi konieczność podzielenia długich dokumentów na potrzeby modelu LLM. Części będące wynikiem podziału dokumentu są zapisywane w postaci osadzeń, co znaczy, że są wektorową reprezentacją danej informacji. Wszystkie te narzędzia wzbogacają wiedzę modeli LLM i zwiększają ich wydajność w takich zadaniach jak odpowiadanie na pytania czy streszczanie.

Dla bazy wektorowej mamy do wyboru wiele integracji, między innymi: Alibaba Cloud OpenSearch, AnalyticDB dla PostgreSQL, bibliotekę Meta AI's Annoy **dla wyszukiwania najbliższego sąsiada**, Cassandra, Chroma, Elasticsearch, **Facebook AI Similarity Search (Faiss)**, MongoDB Atlas Vector Search, PGVector, wyszukiwanie podobnego wektora dla Postgres, Pinecone, scikit-learn (SKLearnVectorStore dla wyszukiwania *k* najbliższych sąsiadów) i wiele więcej. Odkryjemy je w rozdziale 5. „Tworzenie robota konwersacyjnego na kształt ChatGPT".

Uwaga

Choć w późniejszych rozdziałach zagłębimy się w szczegóły kilku wzorców użycia i przypadków użycia komponentów LangChain, w podanych niżej źródłach wiedzy znajdziesz bezcenne informacje o komponentach frameworku LangChain i dowiesz się, jak można je łączyć ze sobą w potoki.

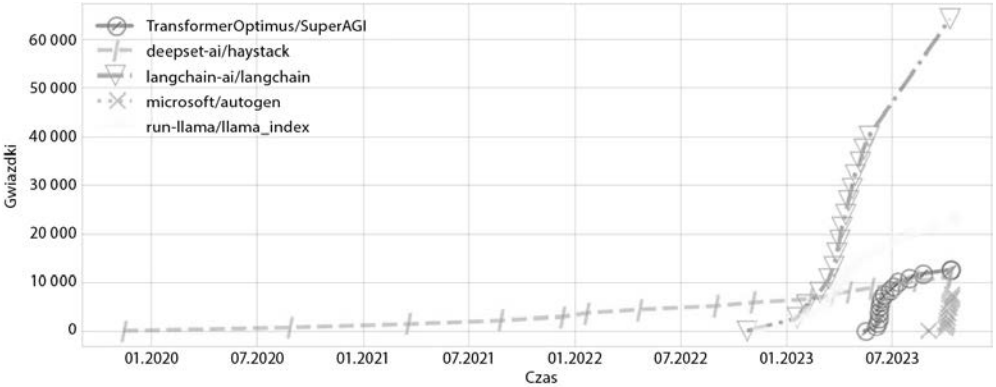
Pełna baza wiedzy na temat dostępnych modułów znajduje się w dokumentacji API na stronie *https://api.python.langchain.com/en/latest/experimental_api_reference.html*. Z kolei na stronie *https://python.langchain.com/docs/use_cases/* znajdziesz setki przykładów rzeczywistego użycia różnych modułów.

LangChain nie jest jedynym frameworkiem, ale przekonasz się, że jest czołowym frameworkiem do pracy z modelami językowymi i ma najwięcej funkcji.

Porównanie LangChain z innymi frameworkami

Frameworki do tworzenia aplikacji LLM powstały z myślą o zapewnieniu specjalnego zestawu narzędzi w celu ujarzmienia modeli LLM i ich niesamowitych zdolności rozwiązywania złożonych problemów. Pojawiło się kilka bibliotek spełniających wymagania co do skutecznego łączenia generatywnych modeli AI z innymi narzędziami, by można było budować aplikacje oparte na modelach LLM.

Poza LangChain jest jeszcze klika innych frameworków z otwartym źródłem do tworzenia dynamicznych aplikacji LLM. Wszystkie mają swoje zalety w zakresie tworzenia nowoczesnych aplikacji. Wykres z rysunku 2.11 pokazuje, jak w czasie kształtowała się popularność poszczególnych z nich (źródło: *GitHub star history*, *https://star-history.com/*).



Rysunek 2.11. Porównanie popularności różnych frameworków w Pythonie

Dla każdego z widocznych na wykresie projektów mamy podaną liczbę gwiazdek i widać wyraźnie, jak ta liczba dla poszczególnych z nich zmieniała się w czasie. Haystack jest najstarszym z porównywanych frameworków. Jego początki sięgają 2020 roku (z początku 2020 roku pochodzą pierwsze zmiany w repozytorium). Jednak liczba gwiazdek na GitHubie pokazuje, że jest najmniej popularny. LangChain, LlamaIndex (wcześniej GPTIndex) i SuperAGI powstały pod koniec 2022 roku lub na początku 2023. Spośród nich jedynie popularność LangChain znacząco wzrosła, pozostałe frameworki szybko straciły na popularności. Projekt AutoGen został opublikowany niedawno przez firmę Microsoft i już zyskał pewne zainteresowanie. W tej książce zobaczymy wiele funkcjonalności frameworku LangChain i odkryjemy jego cechy i wówczas sam się przekonasz, z czego wynika tak duża jego popularność.

Framework LlamaIndex bardziej skupia się na zaawansowanym pozyskiwaniu danych niż na szerszym aspekcie tworzenia aplikacji opartych na modelach LLM. Podobnie Haystack skupia się na tworzeniu systemów wyszukujących na szeroką skalę i jego komponenty zostały zaprojektowane specjalnie dla skalowalnego pobierania informacji za pomocą modułów pozyskujących dane, odczytujących dane i innych modułów obsługujących połączonych z indeksowaniem semantycznym poprzez wstępnie wytrenowane modele.

LangChain wyróżnia się tym, że z łatwością łączy modele LLM w łańcuchy z użyciem agentów delegujących tym modelom różne zadania. LangChain wykorzystuje optymalizacje monitów oraz pobieranie/generowanie informacji z uwzględnieniem kontekstu. Jednak to przede wszystkim dzięki swojej modułowej naturze i ogromnej kolekcji dostępnych narzędzi LangChain jest numerem jeden wśród frameworków do implementowania złożonej logiki biznesowej.

SuperAGI ma podobne funkcje do frameworku LangChain. Ma nawet własne repozytorium z narzędziami i agentami. Jednak nie jest tak rozbudowany i dobrze wspierany jak LangChain.

AutoGen ułatwia budowanie złożonych przepływów pracy zasilanych modelami LLM, zarządzanie nimi i ich optymalizowanie. Wyróżnia się tym, że zapewnia agenty konwersacyjne, które można dostosować do własnych potrzeb i które automatycznie koordynują interakcje między różnymi modelami LLM, ludźmi i narzędziami poprzez zautomatyzowany czat. AutoGen upraszcza definicje i interakcje agenta, by automatycznie tworzyć optymalne przepływy pracy oparte na modelach LLM.

Nie wspomniałem o AutoGPT (i podobnych mu narzędziach jak AutoLlama) — rekurencyjnej aplikacji, która rozkłada zadania na części pierwsze — ponieważ jej możliwości rozumowania, oparte na informacjach zwrotnych od człowieka i modelu LLM, są bardzo ograniczone w porównaniu z LangChain. W rezultacie często się zapętla i bez przerwy powtarza te same kroki. Pominąłem również kilka bibliotek, które oscylują wokół inżynierii monitów, na przykład Promptify.

Są jeszcze inne frameworki do tworzenia aplikacji opartych na modelach LLM, w których to narzędziach możesz programować za pomocą takich języków jak Rust, JavaScript, Ruby i Java. Na przykład framework Dust, napisany w języku Rust, skupia się na projektowaniu aplikacji LLM i ich wdrażaniu.

Frameworki takie jak LangChain w celu obniżenia progu wejścia dla programistów zapewniają przewodniki, konwencje i wstępnie przygotowane moduły. Jednakże podstawowa wiedza jest niezbędna, by uniknąć porażek i wyciągnąć jak najwięcej z modeli LLM. Warto zainwestować w swoją edukację. To się opłaci, gdy będziesz umiał dostarczać aplikacje, które mają wiele funkcji i działają w odpowiedzialny sposób.

Podsumowanie

Modele LLM wytwarzają przekonujące treści, ale mają poważne ograniczenia w kwestii rozumowania, zakresu wiedzy i dostępu do narzędzi. Framework LangChain ułatwia budowanie zaawansowanych aplikacji zasilanych modelami LLM, które stawiają czoło ograniczeniom dużych modeli językowych. LangChain zapewnia programistom modułowe komponenty wielokrotnego użycia, między innymi łańcuchy do budowy potoków czy agenty do interakcji o określonym celu. Te komponenty razem są dobrze do siebie dopasowane i razem tworzą aplikacje LLM z szerokimi możliwościami.

Jak mieliśmy się okazję dowiedzieć w tym rozdziale, łańcuchy umożliwiają tworzenie sekwencji wysyłania monitów do modeli LLM, baz danych, API itp., by wykonać wszystkie kroki złożonych przepływów pracy. Agenty wykorzystują łańcuchy do wykonywania działań opartych na obserwacjach, dzięki czemu są w stanie zarządzać dynamicznymi aplikacjami. W pamięci można przechowywać informacje o poszczególnych działaniach komponentów dla zachowania stanu. Te wszystkie koncepcje pozwalają programistom obejść ograniczenia poszczególnych modeli LLM, ponieważ dzięki temu modele te mogą integrować zewnętrzne dane, akcje i kontekst. Innymi słowy, LangChain upraszcza pracę, gdyż dostarcza klocki, które można dostosowywać do określonych zadań i potrzeb.

W następnych rozdziałach poszerzymy naszą podstawową wiedzę o LangChain i będziemy tworzyć wielofunkcyjne aplikacje, które mogą mieć rzeczywiste zastosowania. Zaimplementujemy agenta konwersacyjnego przez połączenie modelu LLM z bazami wiedzy i zaawansowanymi algorytmami rozumowania. Poprzez wykorzystanie możliwości frameworku LangChain programiści mogą uwolnić cały potencjał modeli LLM i tworzyć nowatorskie oprogramowanie oparte na sztucznej inteligencji. W następnym rozdziale zaimplementujemy swoją pierwszą aplikację z użyciem LangChain!

Pytania

Sprawdź, czy potrafisz odpowiedzieć na pytania podane poniżej. Jeśli masz problem z odpowiedzią na któreś z pytań, wróć do odpowiednich podrozdziałów lub punktów.

1. Wymień ograniczenia modeli LLM.
2. Czym są papugi stochastyczne?
3. Podaj zastosowania modeli LLM.
4. Co to jest LangChain i dlaczego powinniśmy go używać?
5. Jakie są kluczowe funkcjonalności LangChain?
6. Czym w LangChain są łańcuchy?
7. Czym jest agent?
8. Czym jest pamięć i dlaczego jej potrzebujemy?
9. Wymień rodzaje narzędzi dostępnych w LangChain.
10. Jak działa LangChain?

Okiełznaj imponującą moc generatywnej AI!

Generatywne modele językowe, takie jak ChatGPT, zrewolucjonizowały techniki przetwarzania informacji: tekstu, obrazów i sekwencji wideo. Stosowanie uczenia głębokiego i modeli LLM już teraz mocno wpływa na przedsiębiorstwa i społeczeństwa. Potencjał tych innowacji jest ogromny: generatywne modele językowe stały się wiodącym trendem w tworzeniu aplikacji i analizie danych.

Dzięki tej przystępnej książce, przeznaczonej dla programistów i badaczy, zrozumiesz podstawy techniczne modeli LLM. Dowiesz się, do czego można je zastosować, i odkryjesz elegancję ich architektury. Nauczysz się praktycznego korzystania z frameworka LangChain, zaprojektowanego do tworzenia responsywnych aplikacji. Dowiesz się, jak dostrajać model, jak zadawać mu pytania, poznasz także sprawdzone metody wdrażania i monitorowania środowisk produkcyjnych, dzięki czemu łatwo zbudujesz narzędzia do pisania, zaawansowane roboty konwersacyjne czy nowatorskie pomoce dla programistów. Liczne praktyczne przykłady i fragmenty kodu ułatwią Ci nie tylko przyswojenie podstaw, ale także używanie modeli LLM w innowacyjny i odpowiedzialny sposób.

Najciekawsze zagadnienia:

- mocne strony i ograniczenia LLM
- zastosowanie frameworka LangChain do tworzenia aplikacji AI
- modele transformatora i mechanizmy uwagi
- automatyzacja analizy danych i wizualizacji za pomocą języka Python i biblioteki pandas
- dostrajanie i wdrażanie modeli LLM z użyciem LangChain, a także strategie oceniające
- zapobieganie wyciekom danych

Dr Ben Auffarth jest doświadczonym liderem w dziedzinie analizy danych. Zajmował się symulacją aktywności mózgu na superkomputerach z 64 tysiącami rdzeni, zaprojektował i przeprowadził wiele eksperymentów, trenował sieci neuronowe na milionach dokumentów.

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 978-83-289-1436-0
9 788328 914360
Cena: 79,00 zł
packt