Architektura aplikacji dla generatywnych aplikacji AI

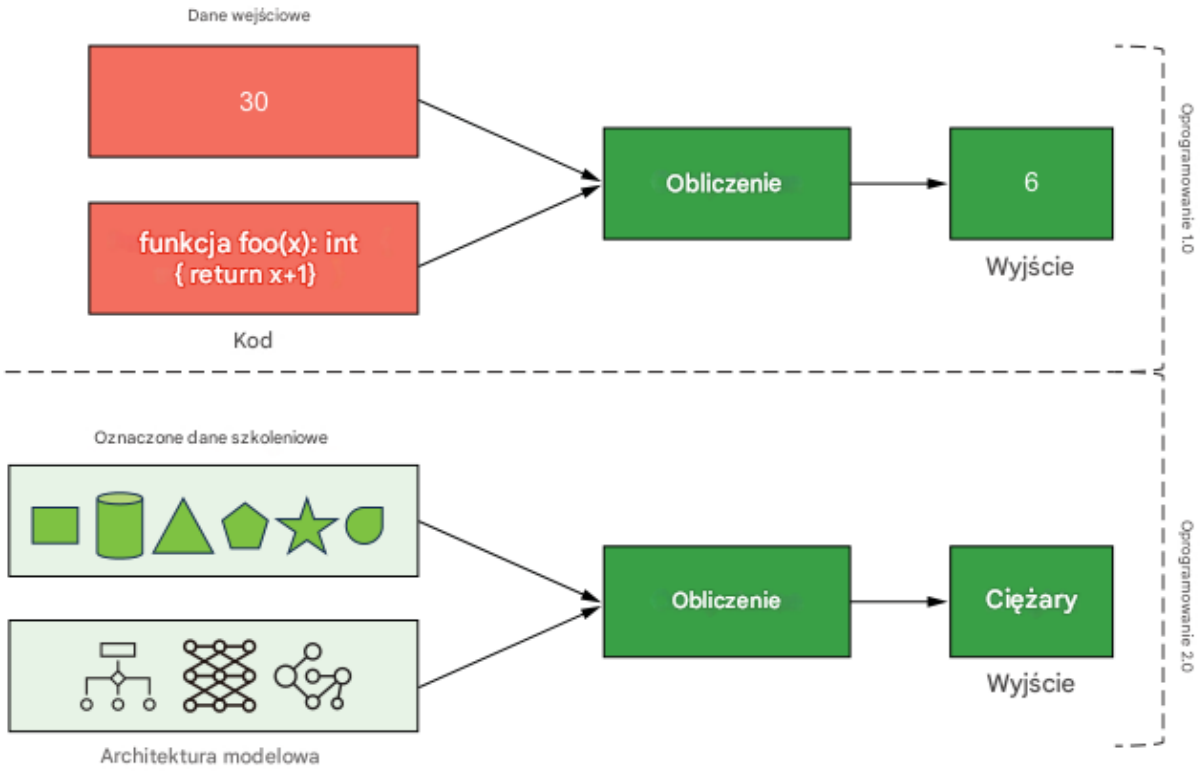
Krajobraz architektury korporacyjnej nieustannie się zmienia, nieubłaganie zmierzając w kierunku systemów bardziej autonomicznych – inteligentnych, samozarządzających aplikacji, które potrafią uczyć się na podstawie interakcji i adaptować się w czasie rzeczywistym. Co więcej, rosnąca cyfryzacja napędza cyfrową transformację sztucznej inteligencji (AI). Ten ciągły postęp podkreśla transformacyjną erę w technologii korporacyjnej, gotową na nowo zdefiniować samą naturę tworzenia i wdrażania oprogramowania. Oczywiście, jest to raczej ideał. Jednak większość przedsiębiorstw wciąż ma niewielkie doświadczenie w zakresie aplikacji wykorzystujących sztuczną inteligencję (AI), a generatywna AI jest wciąż w bardzo wczesnej fazie rozwoju. W tym rozdziale omówimy, w jaki sposób standardy architektury aplikacji korporacyjnych i najlepsze praktyki muszą dostosować się do pojawiających się technologii i przypadków użycia generatywnej AI. Rozdział wprowadza koncepcję stosu aplikacji GenAI jako koncepcyjnej architektury referencyjnej do tworzenia generatywnych aplikacji AI, a także opisuje jego główne komponenty i sposób, w jaki generatywna AI wpisuje się w szerszą architekturę przedsiębiorstwa. Stos aplikacji GenAI to ewolucja architektury aplikacji chmurowych, z przesunięciem w kierunku architektur skoncentrowanych na danych i opartych na AI. Ten rozdział rozpoczyna się od omówienia, co obejmuje nowy stos aplikacji GenAI, omawiając szczegółowo każdą sekcję, a na koniec łącząc wszystkie koncepcje w praktyczne przykłady, które czynią go realnym i użytecznym. W miarę jak będziesz poznawać ten stos, skonsolidujemy różne aspekty architektury opisanej w poprzednich rozdziałach. Warto zauważyć, że pomimo dużej zmiany, generatywna sztuczna inteligencja nie wymaga całkowicie nowej architektury, lecz opiera się na istniejącej, rozproszonej architekturze chmurowej. Ta cecha pozwala nam wykorzystać istniejące najlepsze praktyki i zasady architektury, aby włączyć nowe paradygmaty związane z GenAI. Zacznijmy od zidentyfikowania aktualizacji w architekturze aplikacji korporacyjnych.

Generatywna sztuczna inteligencja: Architektura aplikacji

W ciągu ostatnich kilku lat architektura aplikacji korporacyjnych przeszła znaczącą ewolucję, przechodząc przez kilka etapów transformacji, aby sprostać rosnącym wymaganiom w zakresie zwinności biznesowej, skalowalności i inteligencji. Początkowo przedsiębiorstwa działały w oparciu o systemy monolityczne, czyli solidne, ale nieelastyczne struktury ze ściśle powiązanymi komponentami, co utrudniało wprowadzanie zmian i miało szeroki zasięg. Systemy te utorowały drogę dla przetwarzania korporacyjnego, ale nie były dostosowane do szybkiej ewolucji potrzeb biznesowych. Rozpowszechnienie przetwarzania w chmurze i architektur natywnych dla chmury doprowadziło do powstania narzędzi do konteneryzacji i orkiestracji, które uprościły wdrażanie i zarządzanie aplikacjami w różnych środowiskach. Jednocześnie napływ danych doprowadził do powstania architektur zorientowanych na dane, które priorytetowo traktują przetwarzanie i analizę danych jako kluczowe czynniki napędzające działalność biznesową. Ewolucję architektury aplikacji korporacyjnych dla generatywnej sztucznej inteligencji można postrzegać jako przejście od tradycyjnego tworzenia oprogramowania do syntezy oprogramowania opartej na danych. W tradycyjnym paradygmacie inżynierowie oprogramowania piszą kod w celu implementacji określonych funkcjonalności i logiki, używając frameworków i bibliotek, które abstrahują szczegóły niskiego poziomu. W paradygmacie generatywnej sztucznej inteligencji programiści dostarczają dane i specyfikacje wysokiego poziomu oraz wykorzystują duże modele językowe (LLM) do generowania kodu spełniającego pożądane wymagania i ograniczenia. Następujące dwie kluczowe koncepcje umożliwiły tę zmianę paradygmatu: Oprogramowanie 2.0 i budowanie na współpilotach.

Oprogramowanie 2.0

Oprogramowanie 2.0 to termin ukuty przez Andreja Karpathyego, opisujący trend zastępowania ręcznie tworzonego kodu wyuczonymi sieciami neuronowymi. Oprogramowanie 2.0 wykorzystuje postęp w dziedzinie sztucznej inteligencji, taki jak przetwarzanie języka naturalnego (NLP), widzenie komputerowe i uczenie przez wzmacnianie, do tworzenia komponentów oprogramowania, które mogą uczyć się na podstawie danych, adaptować się do nowych sytuacji i naturalnie oddziaływać na ludzi. Ostatnio przeszliśmy od pisania kodu i zarządzania wyraźnymi instrukcjami dla osiągnięcia pożądanego celu do bardziej abstrakcyjnego podejścia. Programiści trenują modele na dużych zbiorach danych, zamiast pisać wyraźne instrukcje lub reguły w języku programowania. Oprogramowanie 2.0 zmniejsza również potrzebę ręcznego debugowania, testowania i konserwacji, ponieważ sieci neuronowe mogą samoczynnie korygować i ulepszać się z czasem .



Pozwala to modelom samodzielnie uczyć się reguł lub wzorców. Algorytmy i modele są tworzone tak, aby uczyć się na podstawie danych, podejmować decyzje i udoskonalać się z czasem, skutecznie tworząc oprogramowanie. Ta zmiana paradygmatu przekształciła rolę sztucznej inteligencji z narzędzia wspomagającego w fundamentalny element architektury systemu.

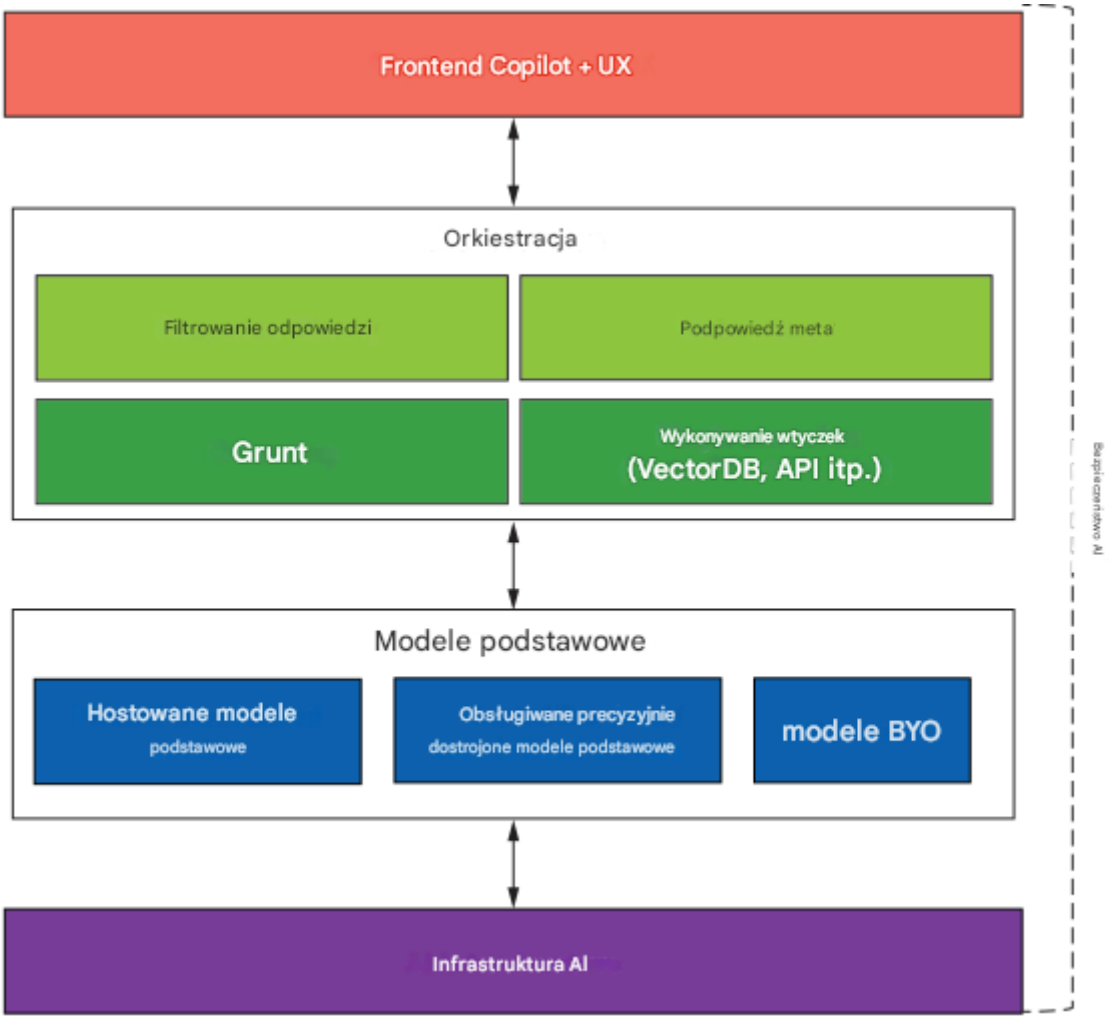
Era drugich pilotów

Kolejną kluczową koncepcją, która ułatwiła ewolucję architektury aplikacji korporacyjnych dla generatywnej sztucznej inteligencji, są „drugi piloci" – koncepcja pierwotnie zaproponowana przez Microsoft. „Drugi piloci" mają na celu wspomaganie ludzi, ich możliwości i kreatywności. Używając analogii do samolotu, jeśli jesteśmy ludźmi, to jesteśmy pilotami; zamiast, gdy sztuczna inteligencja działa na autopilocie i nie mamy kontroli ani wpływu na jej funkcjonowanie, ta nowa sztuczna inteligencja pełni rolę „drugich pilotów", którzy pomagają nam przejąć obciążenie poznawcze i część żmudnej pracy. Nadal jednak to my, jako pilot, sprawujemy kontrolę. Stos „Drugi pilot" to platforma do tworzenia aplikacji sztucznej inteligencji i „drugich pilotów", które wykorzystują modele LLM do rozumienia i generowania języka naturalnego oraz kodu. „Drugi pilot" to inteligentni asystenci, którzy mogą pomagać użytkownikom w złożonych zadaniach poznawczych, takich jak pisanie, kodowanie,

wyszukiwanie czy rozumowanie. Microsoft opracował szereg „drugich pilotów” dla różnych domen i platform, takich jak GitHub Copilot, Bing Chat, Dynamics 365 Copilot i Windows Copilot. Możesz również zbudować własnego Copilota, korzystając ze stosu Copilot i narzędzi, takich jak Azure OpenAI, Copilot Studio i biblioteka Teams AI. Copiloty można również zintegrować z istniejącymi narzędziami i platformami, takimi jak GitHub, Visual Studio Code i Jupyter Notebook, aby zwiększyć produktywność i kreatywność programistów. Copiloty opierają się na koncepcji oprogramowania 2.0, gdzie wykorzystują modele LLM do generowania kodu z opisów w języku naturalnym, zamiast polegać na kodzie pisanym ręcznie. Należy je jednak postrzegać jako stos aplikacji GenAI, podobny do stosu LAMP dla tworzenia stron internetowych. LAMP to akronim od składników stosu: Linux (system operacyjny); Apache (serwer WWW); MySQL (baza danych); oraz PHP, Perl lub Python (język programowania). Copiloty stanowią użyteczny model dla przedsiębiorstw projektujących architekturę korporacyjną generatywnych aplikacji AI, ponieważ oferują szereg korzyści (np. szybsze i prostsze tworzenie oprogramowania, większą kreatywność i możliwość testowania oraz lepszą współpracę i uczenie się, umożliwiając przedsiębiorstwom testowanie nowych koncepcji i możliwości lub tworzenie oryginalnych rozwiązań trudnych problemów). Rozwińmy istotę stosu Copilot, aby uczynić go bardziej istotnym i realnym w praktyce.

Generative AI: Stos aplikacji

Architektura Copilots składa się z kilku warstw i komponentów, które współpracują ze sobą, zapewniając płynne i wydajne środowisko użytkownika, jak przedstawiono na rysunku 2.



Zaczniemy od podstaw, szczegółowo analizując każdą warstwę i komponent, i dowiemy się, jak one ze sobą współdziałają. Warstwa infrastruktury AI to warstwa fundamentowa, która napędza wszystko i hostuje podstawowe modele AI oraz zasoby obliczeniowe. Obejmuje ona sprzęt, oprogramowanie i usługi, które umożliwiają tworzenie i wdrażanie aplikacji AI i są często optymalizowane pod kątem obciążeń AI. Obejmuje to również masowo skalowalne rozproszone obliczenia o wysokiej wydajności (HPC), niezbędne do trenowania podstawowych modeli fundamentowych. Warstwa modelu fundamentowego obejmuje zakres obsługiwanych modeli, od hostowanych modeli fundamentowych po model, który trenujesz i chcesz wdrożyć. Hostowane modele fundamentowe to duże, wstępnie wytrenowane modele, takie jak LLM i inne (modele wizji i mowy), a także nowsze małe modele językowe (SLM), które można wykorzystać do wnioskowania; modele te mogą być zamknięte lub otwarte. Niektóre modele można dodatkowo dostosować do konkretnych zadań lub domen. Modele te są hostowane i zarządzane w warstwie infrastruktury AI, co zapewnia wysoką wydajność i dostępność. Użytkownicy mogą wybierać spośród różnych hostowanych modeli bazowych, w zależności od swoich potrzeb i preferencji. Warstwa orkiestracji zarządza interakcjami między różnymi komponentami architektury, zapewniając płynne działanie i koordynację. Odpowiada ona za kluczowe funkcje, takie jak alokacja zadań, zarządzanie zasobami i optymalizacja przepływu pracy:

* Komponent filtrowania odpowiedzi wykorzystuje zestaw komponentów inżynierii komunikatów; w tym przypadku komunikaty i odpowiedzi są analizowane, filtrowane i optymalizowane w celu generowania bezpiecznych wyników.

* Komunikat systemowy może również dostarczać dodatkowych informacji lub ograniczeń, które model AI powinien przestrzegać. Użytkownik może wyrazić komunikat systemowy za pomocą prostej składni lub system może go automatycznie wygenerować.

* Uziemienie to implementacja generacji rozszerzonej o pobieranie (RAG) i odnosi się do procesu kontekstualizacji odpowiedzi generowanych przez model AI. Uziemienie zapewnia, że wyniki są poprawne składniowo, semantycznie znaczące i istotne w danym kontekście lub domenie. Używamy wtyczek do pobierania danych z różnych systemów przedsiębiorstwa.

* Warstwa wykonania wtyczki uruchamia wtyczki, które dodają więcej funkcji do podstawowego modelu AI. Wtyczki to oddzielne i wielokrotnego użytku części, które mogą wykonywać różne czynności

* Komponent filtrowania odpowiedzi wykorzystuje zestaw komponentów inżynierii komunikatów; w tym przypadku komunikaty i odpowiedzi są analizowane, filtrowane i optymalizowane w celu generowania bezpiecznych wyników.

* Komunikat systemowy może również dostarczać dodatkowych informacji lub ograniczeń, które model AI powinien przestrzegać. Użytkownik może wyrazić komunikat systemowy za pomocą prostej składni lub system może go automatycznie wygenerować.

* Uziemienie to implementacja generacji rozszerzonej o pobieranie (RAG) i odnosi się do procesu kontekstualizacji odpowiedzi generowanych przez model AI. Uziemienie zapewnia, że wyniki są poprawne składniowo, semantycznie znaczące i adekwatne do danego kontekstu lub domeny. Używamy wtyczek do pobierania danych z różnych systemów przedsiębiorstwa.

* Warstwa wykonania wtyczki uruchamia wtyczki, które dodają więcej funkcji do podstawowego modelu AI. Wtyczki to oddzielne i wielokrotnego użytku części, które mogą wykonywać różne czynności, takie jak przetwarzanie danych, formatowanie, walidacja lub transformacja. Jest to bardzo ważne przy pobieraniu danych w celu tworzenia osadzeń oraz stosowaniu wektorowych baz danych i indeksów, gdy w naszych rozwiązaniach stosujemy RAG.

Warstwa UX to interfejs umożliwiający użytkownikom korzystanie z Copilota. Jest on łatwy w obsłudze i oferuje zaawansowane narzędzia do pracy z funkcjami sztucznej inteligencji (AI). Dokładny charakter działania UX zależy od tego, z którym aspektem aplikacji i przepływem pracy łączy się Copilot. Załóżmy na przykład, że Copilot jest częścią pakietu Microsoft Office 365. Sposób działania UX w programie Word różni się od tego w programie PowerPoint i innych aplikacjach, takich jak GitHub Copilot, jak już wcześniej omówiliśmy. Wreszcie, wszystko to odbywa się z uwzględnieniem bezpieczeństwa sztucznej inteligencji (AI), głównego elementu odpowiedzialnej sztucznej inteligencji (AI), zapewniającego etyczne i odpowiedzialne korzystanie z technologii. Bezpieczeństwo sztucznej inteligencji (AI) obejmuje różne metody i zasady, które wyjaśnimy w dalszej części.

Integracja stosu GenAI

Aby zintegrować stos GenAI z aplikacją korporacyjną, wymagane jest strategiczne i techniczne podejście. Zaczyna się ono od poznania wymagań i wyzwań biznesowych, którym można sprostać za pomocą GenAI, zwłaszcza LLM. Obejmuje to powiązanie konkretnych przypadków użycia z możliwościami tych technologii AI, koncentrując się na obszarach, w których mogą one przynieść znaczną wartość, takich jak usprawnienie złożonych przepływów pracy, ulepszenie analityki danych lub ułatwienie interakcji z klientami. Mając na uwadze te przypadki użycia, kolejnym krokiem jest stworzenie architektury integracji, która dopasuje stos AI do granic obecnego systemu przedsiębiorstwa, wykorzystując architekturę zorientowaną na usługi (SOA) lub podejście mikrousługowe w celu zapewnienia elastyczności. Utworzenie bezpiecznych, skalowalnych i łatwych w utrzymaniu interfejsów API jest ważne dla ułatwienia komunikacji między aplikacją a usługami AI, które będą podstawą integracji. Konfiguracja infrastruktury AI to ważny etap, na którym polityka organizacji i wrażliwość danych decydują o wyborze rozwiązania lokalnego, w chmurze czy hybrydowego. Infrastruktura wymaga sprzętu i opcji przechowywania danych, aby spełnić wymagania danego przypadku użycia. Solidny potok danych jest również niezbędny do efektywnego wnioskowania modeli, zwłaszcza w przypadku korzystania z RAG. Dzięki GenAI programiści mogą korzystać z istniejących modeli z usług AI w chmurze lub uruchamiać modele na swoich serwerach. Programiści mogą budować lub dostosowywać modele specyficzne dla danej domeny, gdy wymagane są niestandardowe rozwiązania, zapewniając sobie szkolenie na dokładnych i trafnych zestawach danych oraz dodając metody ciągłego uczenia się w celu wzbogacenia modelu o nowe dane. Aby zapewnić odpowiedzialne wdrażanie AI (RAI), bezpieczeństwo i zgodność z przepisami o ochronie danych, stosowane są systemy filtrowania odpowiedzi, zapobiegające tworzeniu nieodpowiednich treści i naruszaniu przepisów. Doświadczenie użytkownika opiera się na tym projekcie UX, który umożliwia użytkownikom interakcję ze stosem AI. Proces projektowania jest powtarzany, uwzględniając opinie użytkowników, aby efektywnie zaspokoić potrzeby przedsiębiorstwa. System umożliwia integrację z aplikacjami innych firm lub niestandardowe rozszerzenia poprzez bezpieczną architekturę wtyczek, która może je uruchamiać bez wpływu na niezawodność aplikacji. Warstwa orkiestracji obsługuje interakcje między różnymi komponentami AI, zapewniając, że system może dostosowywać się do różnych poziomów zapotrzebowania. Wdrożenie jest zautomatyzowane, aby zapewnić spójne i niezawodne aktualizacje stosu AI, a potoki CI/CD umożliwiają ciągłą integrację i dostarczanie bez zakłócania istniejących funkcjonalności. Wydajność i stan stosu AI są stale monitorowane, a kompleksowe systemy rejestrowania i alertów powiadamiają o wszelkich problemach. Wreszcie, pomyślne wdrożenie i uruchomienie stosu AI zależy od dobrze udokumentowanych wytycznych i szkoleń dla programistów i użytkowników końcowych, które zapewnią im pełne przygotowanie do użytkowania, rozwiązywania problemów i utrzymywania nowego systemu. Każdy etap tego procesu wymaga szczegółowego planowania, współpracy między zespołami i dogłębnej wiedzy technicznej, aby zapewnić płynną i skuteczną integrację z architekturą przedsiębiorstwa. Przyjrzyjmy się bliżej stosowi GenAI.

Zasady architektury GenAI

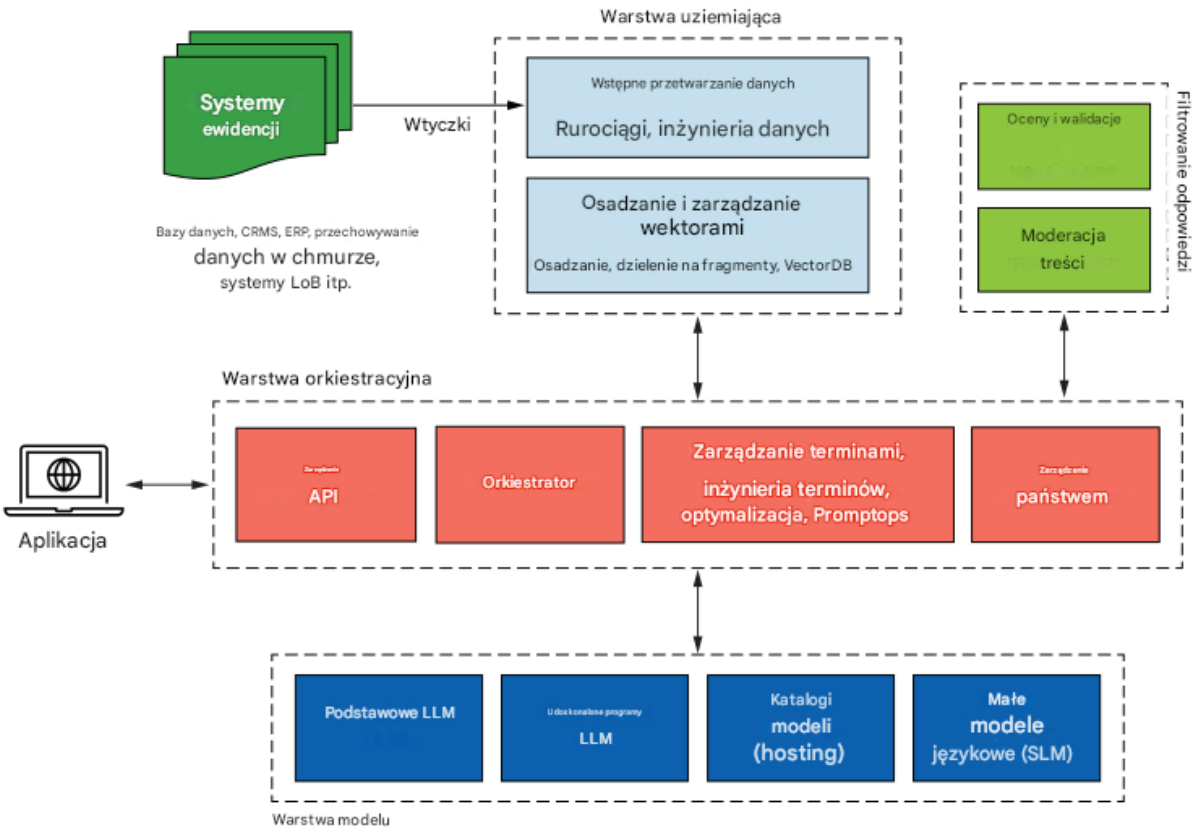
Tworząc aplikacje o znaczeniu krytycznym, przedsiębiorstwa koncentrują się na stworzeniu solidnego, skalowalnego i bezpiecznego systemu. Chociaż tradycyjne zasady architektoniczne pozostają niezmienione, kluczowe dodatkowe aspekty architektoniczne dla generatywnej AI przedstawiono na rysunku 3.

Dostęp do wielu modeli GenAI odbywa się za pośrednictwem interfejsu API, dlatego integracja modelu z API stanowi zasadę architektury, która ułatwia połączenie z API GenAI. Modele i API różnią się sposobem formatowania i przesyłania danych, a także limitami i limitami liczby żądań, które mogą obsłużyć. Dlatego pomocne może być utworzenie warstwy abstrakcji, która może dostosowywać się do zmian w projekcie każdego API. Obejmuje to obsługę żądań i odpowiedzi API oraz zarządzanie limitami i limitami API. Często w tej samej aplikacji używanych jest wiele modeli, aby wybrać odpowiedni model do każdej sytuacji. Posiadanie warstwy abstrakcji może pomóc w ochronie projektu każdego API przed zmianami. Zasadniczo skalowalność i wydajność pomagają aplikacji radzić sobie z elastyczną skalą i zmieniającym się obciążeniem w miarę jego wzrostu i spadku. Wiąże się to z wyborem odpowiedniej infrastruktury chmurowej, równoważeniem obciążenia i potencjalnym wykorzystaniem przetwarzania asynchronicznego do zarządzania intensywnymi zadaniami. Ponadto, wykorzystanie konteneryzacji i architektury mikrousług może pomóc zarówno w zapewnieniu skalowalności, jak i wydajności. Hosting systemów LLM w centrum danych przedsiębiorstwa nie jest łatwym zadaniem, ponieważ wymaga starannego planowania w celu osiągnięcia skalowalności i wydajności. Należy wybrać odpowiednią architekturę LLM, porównując alternatywy open source i zastrzeżone, które są zgodne z celami biznesowymi. Usprawniony, kompleksowy proces przetwarzania danych ma kluczowe znaczenie dla płynnego działania, wykorzystując ramy orkiestracji do zarządzania przepływem pracy. Infrastruktura powinna być solidna pod kątem optymalizacji GPU i uproszczonego zarządzania infrastrukturą. Podczas wdrażania należy stosować LLMOps zgodnie z najlepszymi praktykami, a także zapewnić ciągły monitoring wydajności. Skalowalność należy zapewnić poprzez równoważenie

obciążenia i automatyczne skalowanie. Dane i modele powinny być zabezpieczone szyfrowaniem i kontrolą dostępu, a także należy przestrzegać przepisów branżowych. To kompleksowe podejście gwarantuje, że systemy LLM mogą sprawnie i niezawodnie obsługiwać wielu klientów wewnętrznych. Oczywiście wiąże się to ze znacznymi i ciągłymi inwestycjami w nakłady inwestycyjne i wiedzę techniczną. Ze względu na wrażliwą naturę danych, kluczowe jest wdrożenie silnych środków ochrony prywatności i bezpieczeństwa danych, które obejmują szyfrowanie danych zarówno w tranzycie, jak i w spoczynku, zarządzanie kontrolą dostępu oraz zapewnienie zgodności z przepisami takimi jak RODO czy HIPAA. Ponadto ważne jest posiadanie strategii minimalizacji danych, w ramach której gromadzone i przetwarzane są tylko niezbędne dane, a audyty bezpieczeństwa i testy penetracyjne powinny być regularnie przeprowadzane w celu proaktywnego identyfikowania i usuwania luk w zabezpieczeniach. Niektórzy dostawcy usług w chmurze, tacy jak Azure, oferują solidne systemy wsparcia dla przedsiębiorstw i rozwiązania zapewniające zgodność z przepisami. Obsługa błędów i monitorowanie nie stanowią nowej zasady architektury; w przypadku systemów rozproszonych, jeśli nie planujesz awarii, planujesz awarię. Wykorzystaj skuteczną obsługę błędów i monitorowanie, aby sprawdzić kondycję aplikacji GenAI. Oznacza to rejestrowanie błędów, tworzenie alertów dotyczących anomalii oraz posiadanie planu radzenia sobie z przestojami lub limitami API, w tym stosowanie strategii automatycznego odzyskiwania, takich jak mechanizmy awaryjne, w celu zapewnienia wysokiej dostępności. Rozproszone śledzenie jest niezbędne dla złożonych architektur opartych na mikrousługach, aby lepiej śledzić problemy. LLM ewoluują pod względem kosztów i walut. Wzrost wykorzystania LLM może prowadzić do nieoczekiwanych wydatków. Aby kontrolować koszty, optymalizuj wywołania API i stosuj strategie buforowania. Wprowadź mechanizmy alertów budżetowych i prognozowania kosztów, aby uniknąć niespodzianek. Projekt UX GenAI koncentruje się na tym, jak użytkownicy wchodzą w interakcję z modelami GenAI. Będzie się to różnić w zależności od typu modelu; w przypadku użycia opartego na języku z wykorzystaniem LLM, projekt UX będzie się znacznie różnić od przypadku użycia opartego na obrazach, w którym używałbyś Stable Diffusion lub DALL-E. Obejmuje to projektowanie intuicyjnych interfejsów, dostarczanie pomocnych podpowiedzi i zapewnienie, że odpowiedzi modelu są zgodne z oczekiwaniami użytkownika. W pewnym sensie nie wszystko powinno być prostym chatbotem, ale powinno rozszerzać i ulepszać doświadczenie w oparciu o zadanie i intencję. Weź pod uwagę etyczne, stronnicze i prawne implikacje aplikacji GenAI, zwłaszcza w przypadku korzystania z LLM. Ograniczaj uprzedzenia i zapobiegaj szkodliwym stereotypom. Zrozum konsekwencje prawne w opiece zdrowotnej, finansach lub prawie. Przestrzegaj odpowiednich przepisów i standardów branżowych.

Architektura aplikacji GenAI: Widok szczegółowy

Na podstawie diagramu architektury wysokiego poziomu i wchodząc w szczegóły, rysunek 10. ilustruje ogólną strukturę stosu aplikacji GenAI.



Chociaż wiele z tych elementów wykorzystaliśmy już w poprzednich rozdziałach, to jest to nasze pierwsze całościowe spojrzenie na nią. Istnieje sześć szerokich kategorii tworzących różne komponenty, które tworzą stos aplikacji GenAI. Następnie przyjrzymy się bliżej każdej warstwie.

WARSTWA ORKIESTRACJI

Warstwa orkiestracji to centralny komponent integrujący różne usługi oraz zarządzający przepływem danych i monitami. Odpowiada za planowanie zadań, przydzielanie zasobów i odporną obsługę błędów. System zarządzania monitami jest kluczowym elementem tej warstwy, wykorzystując technologię sztucznej inteligencji (AI) do tworzenia monitów, które generują najlepsze możliwe odpowiedzi od LLM. Obejmuje to testy A/B i modele uczenia maszynowego (ML) w celu analizy interakcji użytkowników i optymalizacji monitów w celu zwiększenia zaangażowania i dokładności. Narzędzia orkiestracji, takie jak Kubernetes, mogą zarządzać skonteneryzowanymi mikrousługami i umożliwiać wdrażanie komponentów u dostawców chmury i w środowiskach lokalnych, aby poprawić odporność systemu i jego odporność na błędy.

WARSTWA UZIEMIENIA

Ta warstwa stanowi podstawę aplikacji GenAI, które zajmują się pobieraniem, przechowywaniem, przetwarzaniem i dostarczaniem danych. Musi współpracować z różnymi systemami rekordów, wymagając konektorów do obsługi różnych formatów i protokołów danych. Potoki danych to kanały łączące się z różnymi systemami źródłowymi w celu pobierania danych do stosowania RAG i umożliwienia przedsiębiorstwom korzystania z ich danych. Potoki danych mogą łączyć się z systemem rekordów natywnie za pośrednictwem interfejsów API (jeśli są obsługiwane) lub za pomocą różnych wtyczek. Potoki danych powinny być budowane z myślą o dużej szybkości i niskim opóźnieniu, z

możliwością obsługi przetwarzania wsadowego i strumieniowego w razie potrzeby. Środowisko wykonawcze wtyczki uwzględnia różne aspekty uwierzytelniania, konfiguracje odświeżania danych itd. Wstępne przetwarzanie danych jest ważne dla konwersji danych do formatu obsługiwanego przez modele LLM. Dlatego ta warstwa obejmuje modele ML do tworzenia osadzenia i wektorowych baz danych, takich jak Redis, o czym mówiliśmy wcześniej w książce, lub inne, takie jak Cosmos DB, Pinecone, Milvus, Qdrant itd. Wykorzystanie rozproszonych platform przetwarzania danych, takich jak Apache Spark lub Azure Fabric, zapewnia skalowalność i odporność na błędy w przetwarzaniu danych.

WARSTWA MODELU

Warstwa modelu musi obsługiwać zróżnicowany zakres modeli, od pionierskich modeli LLM ogólnego przeznaczenia, takich jak GPT-4, po wysoce wyspecjalizowane modele SLM, takie jak Phi-2 i Orca 2 . W dalszej części rozdziału dowiemy się więcej o modelach SLM i zobaczymy przykład użycia Phi-2 jako klasyfikatora. W rezultacie warstwa modelu powinna zapewniać spójny interfejs dostępu do różnych modeli, niezależnie od tego, czy są one hostowane wewnętrznie, czy zewnętrznie. Rozważając hosting modeli, kluczowe jest skalowanie modeli w celu obsługi zróżnicowanych obciążeń, co może wymagać zastosowania technologii takich jak przetwarzanie bezserwerowe, aby dynamicznie przydzielać zasoby. Katalogi modeli pełnią funkcję rejestru i repozytorium, upraszczając wyszukiwanie i zarządzanie modelami. Ta warstwa obejmuje również koncepcję modelu jako platformy, która pozwala programistom rozszerzać i dostosowywać modele, podobnie jak platformy takie jak Salesforce umożliwiają dostosowywanie aplikacji.

FILTROWANIE ODPOWIEDZI

Ta warstwa ma kluczowe znaczenie dla utrzymania zaufania do aplikacji GenAI poprzez zapewnienie kontroli jakości i moderacji treści. Obejmuje ona wykorzystanie klasyfikatorów i narzędzi NLP do weryfikacji wyników pod kątem dokładności, stronniczości i adekwatności. Odpowiedzialne praktyki AI są zintegrowane z tą warstwą, uwzględniając kwestie etyczne i zapewniając zgodność z przepisami, takimi jak RODO o ochronie danych osobowych. System buforowania w tej warstwie poprawia wydajność i umożliwia szybkie wycofywanie danych oraz sprawdzanie ich przydatności. Ciągły monitoring i ocena danych w czasie rzeczywistym zapewniają utrzymanie wskaźnika AIQ przez cały cykl życia aplikacji. Co więcej, warstwa ta uwzględnia również etyczne implikacje technologii GenAI, w tym opracowanie ram etycznego podejmowania decyzji, zapewnienie przejrzystości modelu oraz uwzględnienie uczciwości i inkluzywności w projektowaniu systemów AI.

Dodatkowe zagadnienia dotyczące architektury

Choć architektura opisana wcześniej jest kompleksowa, nie uwzględnia ona następujących dodatkowych zagadnień, które są kluczowe dla wdrożenia produkcyjnego i dobrze rozumiane przez większość przedsiębiorstw:

* Integracje — aplikacje te nie działają samodzielnie i muszą łączyć się z resztą systemu przedsiębiorstwa, co umożliwia płynny transfer danych i usług między systemami wewnętrznymi i zewnętrznymi. Technologie oprogramowania pośredniczącego, takie jak magistrale usług przedsiębiorstwa (ESB) lub bramy API, są wykorzystywane do obsługi komunikacji i konwersji danych między różnymi systemami.

* Bezpieczeństwo — bezpieczeństwo zawsze było kwestią priorytetową i tak samo jest w przypadku GenAI; wszystkie dane w ekosystemie GenAI muszą być chronione przed nieautoryzowanym dostępem i naruszeniami, co wymaga silnych metod uwierzytelniania i autoryzacji, szyfrowania tranzytowego i REST oraz częstych audytów bezpieczeństwa.

* Wdrożenie i skalowanie produkcyjne — w tym artykule skupiono się na strategiach wdrażania aplikacji GenAI w różnych środowiskach, co obejmuje wykorzystanie systemów orkiestracji kontenerów do wdrażania, usług automatycznego skalowania do obsługi obciążeń dynamicznych oraz infrastruktury jako kodu do powtarzalnego i niezawodnego udostępniania zasobów.

Warstwa orkiestracji

Generatywne aplikacje AI wymagają warstwy orkiestratora, która stanowi szkielet i jest kluczowa dla zarządzania złożonymi zadaniami i przepływami pracy. Jest to warstwa pośrednicząca i integracyjna między modelami, korporacyjnymi magazynami danych, innymi komponentami i aplikacjami. Koordynuje ona i zarządza różnymi komponentami i procesami, które umożliwiają generowanie treści przez modele AI w ramach architektury przedsiębiorstwa. Zapewnienie wydajności, skalowalności i niezawodności przepływów pracy z wykorzystaniem modeli LLM w zakresie generowania treści jest kluczowe. Główne obowiązki orkiestratora obejmują zarządzanie przepływami pracy i orkiestrację usług, ale można je rozszerzyć o dodatkowe zadania. Orkiestra składa się z kilku komponentów. Struktury orkiestracji upraszczają zarządzanie i interakcję z modelami LLM, eliminując złożoność generowania monitów, zarządzania zasobami i monitorowania wydajności. Zapewniają interfejs wysokiego poziomu, który pozwala programistom skupić się na tworzeniu aplikacji bez zagłębiania się w techniczne szczegóły interakcji z modelami LLM. Tabela 1 przedstawia kluczowe obowiązki.

Obszar: Opisy

Zarządzanie przepływem pracy: Orchestrator zapewnia, że sekwencja procesów – od pobierania i przetwarzania danych po wnioskowanie o modelu AI i dostarczanie odpowiedzi – jest wykonywana w uporządkowany i wydajny sposób. Obejmuje to zarządzanie stanem w celu koordynowania zależności między zadaniami, obsługę błędów, mechanizmy ponawiania prób oraz dynamiczną alokację zasobów w oparciu o obciążenie zadaniami.

Organizacja usług: Zazwyczaj stosuje się architekturę mikrousług, w której każda usługa odpowiada za dyskretną funkcję w generatywnym procesie AI. Orkiestracja usług polega na zarządzaniu tymi usługami w celu zapewnienia skalowalności, komunikacji i płynnego działania. Ponadto platformy konteneryzacji, takie jak Docker, oraz systemy orkiestracji, takie jak Kubernetes, wdrażają, zarządzają i skalują mikrousługi w różnych środowiskach.

Koordynacja przepływu danych: Zapewnienie prawidłowego przepływu danych przez system, od początkowych źródeł danych do modelu i z powrotem do użytkownika końcowego lub aplikacji. Obejmuje to wstępne przetwarzanie danych wejściowych, zarządzanie kolejkami dla żądań przychodzących oraz kierowanie danych wyjściowych do właściwych miejsc docelowych.

Równoważenie obciążenia i automatyczne skalowanie: Moduły równoważenia obciążenia dystrybuują przychodzące żądania wnioskowania AI do wielu instancji, aby zapobiec sytuacji, w której pojedyncza instancja stanie się wąskim gardłem. Automatyczne skalowanie dostosowuje liczbę aktywnych instancji w oparciu o bieżące obciążenie, zapewniając ekonomiczne wykorzystanie zasobów. Obejmuje to również komponenty zarządzania API do zarządzania limitami przepustowości i wdrażania strategii wycofywania obciążeń produkcyjnych.

Wersjonowanie i wycofywanie modeli: Orkiestracja obejmuje utrzymywanie różnych wersji modeli AI i zarządzanie ich wdrażaniem. Umożliwia szybkie wycofanie do poprzednich wersji, jeśli nowy model wykazuje nieoczekiwane zachowanie lub niską wydajność.

Zarządzanie oknami kontekstowymi modelu: Orkiestracja usprawnia interakcje poprzez efektywne zarządzanie oknami kontekstowymi i liczbą tokenów. Śledzi i dynamicznie dostosowuje historię

konwersacji w ramach limitów tokenów modelu i utrzymuje spójność odpowiedzi, szczególnie w przypadku długich lub złożonych wymian. Najlepsze praktyki obejmują efektywne zarządzanie kontekstem, obsługę przypadków skrajnych, ciągłe monitorowanie wydajności oraz uwzględnianie opinii użytkowników w celu ciągłego wprowadzania ulepszeń.

Te różne komponenty współdziałają, tworząc solidny system orkiestracji, który stanowi fundament udanego wdrożenia i funkcjonowania generatywnej technologii AI w sektorze przedsiębiorstw. Taka orkiestracja jest niezbędna dla złożoności i ciągłych zmian w aplikacjach opartych na AI, aby uniknąć nieefektywności, błędów i awarii systemów.

Korzyści płynące z frameworka orkiestracji

Orkiestratorzy są niezbędni do zarządzania złożonymi systemami napędzającymi generatywne aplikacje AI. Systemy te obejmują różnorodne procesy, które wymagają starannej koordynacji za pomocą narzędzi orkiestracji. Orkiestratorzy upraszczają przepływy pracy i zapewniają wykonywanie zadań w odpowiedniej kolejności, z uwzględnieniem zależności i reguł obsługi błędów. Zapewnia to niezawodny i regularny przepływ operacyjny, w którym kroki preprocesowania, obliczeń i postprocesowania są płynnie połączone, zapewniając jakość danych i spójne generowanie wyników.

Skalowalność to kolejny obszar, w którym orkiestracja ma kluczowe znaczenie. Wraz ze zmiennym zapotrzebowaniem, system dynamicznie dostosowuje alokację zasobów, szczególnie w przypadku obciążeń produkcyjnych. Orkiestrator może zapewnić tę zwinność, wykorzystując różne techniki, takie jak moduły równoważenia obciążenia, które równomiernie rozkładają obciążenia, oraz funkcje automatycznego skalowania, które modulują moc obliczeniową w czasie rzeczywistym. Ta elastyczność spełnia wymagania dotyczące obciążenia i optymalizuje wykorzystanie zasobów, równoważąc wydajność i efektywność kosztową. Orkiestratorzy musieliby zarządzać tym w różnych modelach, a także uwzględniać profile obliczeniowe i kosztowe tych modeli. Orkiestratory oferują scentralizowane zarządzanie i monitorowanie. Stanowią one struktury oferujące pulpity nawigacyjne i narzędzia do monitorowania wykorzystania LLM, identyfikowania wąskich gardeł i rozwiązywania problemów. Zwiększa to niezawodność systemu poprzez monitorowanie kondycji usług, reagowanie na awarie i minimalizowanie przestojów. Orkiestratory mogą wykorzystywać zautomatyzowane procesy odzyskiwania, takie jak ponowne uruchamianie instancji lub ich wymiana, co zapewnia ciągłość działania usług. Domyślny model wdrażania to metoda płatności za rzeczywiste wykorzystanie (pay-as-you-go) dla większości dostawców usług LLM w chmurze. Model ten jest współdzielony z innymi klientami, a żądania przychodzące są kolejkowane i przetwarzane według kolejności zgłoszeń. Jednak w przypadku obciążeń produkcyjnych wymagających lepszego środowiska użytkownika usługa Azure OpenAI oferuje funkcję aprowizowanych jednostek przepływności (PTU). Funkcja ta umożliwia klientom rezerwowanie i wdrażanie jednostek mocy obliczeniowej modelu w celu szybkiego przetwarzania i generowania uzupełnień. Minimalne wdrożenie, przyrosty i moc obliczeniowa każdej jednostki PTU różnią się w zależności od typu i wersji modelu. Orkiestrator zarządza różnymi punktami końcowymi wdrożenia, między standardowym systemem płatności za rzeczywiste użycie (pay-as-you-go) a jednostkami PTU, aby zapewnić optymalną wydajność i opłacalność. Orkiestratorzy odgrywają znaczącą rolę w zwiększaniu produktywności i usprawnianiu operacji, co osiąga się na dwa sposoby. Po pierwsze, zmniejszają potrzebę pisania powtarzalnego kodu dla typowych zadań, takich jak szybkie tworzenie i przetwarzanie wyników, zwiększając tym samym produktywność programistów. Po drugie, automatyzują wdrażanie i zarządzanie usługami, minimalizując w ten sposób ryzyko wystąpienia błędu ludzkiego. Ten zautomatyzowany proces redukuje ręczne obciążenie i zapewnia efektywne wykorzystanie zasobów obliczeniowych, usprawniając operacje produkcyjne. Zarządzanie operacjami zostanie omówione bardziej szczegółowo w dalszej części rozdziału. Zgodność i nadzór to podstawowe wymagania każdego przedsiębiorstwa. Orkiestrator może pomóc w egzekwowaniu zgodności,

określając sposób przetwarzania, przechowywania i wykorzystywania danych w przepływie pracy, co gwarantuje zgodność danych z polityką zarządzania danymi i przepisami dotyczącymi prywatności w przedsiębiorstwie. Utrzymanie zaufania i zgodności z przepisami prawa w operacjach przedsiębiorstwa ma kluczowe znaczenie i można je osiągnąć poprzez przestrzeganie polityk zarządzania danymi i przepisów dotyczących prywatności. Wiele osób zna orkiestratory i frameworki orkiestracji. Chociaż frameworki takie jak Kubernetes, Apache Airflow i MLflow są skutecznymi, ogólnymi narzędziami orkiestracji dla inżynierii oprogramowania i mogą wspierać operacje uczenia maszynowego, nie są one przeznaczone wyłącznie do generatywnych aplikacji AI. Orkiestracja przepływów pracy dla generatywnej AI wymaga głębszego zrozumienia niuansów tych złożonych technologii. Wybór frameworka orkiestracji dla generatywnych aplikacji AI zależy od istniejącego stosu technologicznego, złożoności przepływów pracy oraz specyficznych wymagań. Tabela 2 przedstawia frameworki orkiestracji dostosowane do specyficznych potrzeb generatywnych aplikacji AI. Frameworki te mogą obsługiwać tradycyjne przepływy pracy obliczeniowe, zarządzać stanem, kontekstem i spójnością interakcji oraz są zaprojektowane tak, aby spełniać specyficzne wymagania generatywnej AI.

Nazwa: Uwagi

Semantic Kernel: Semantic Kernel to framework OSS firmy Microsoft, którego celem jest stworzenie ujednoliconej struktury dla wyszukiwania semantycznego i generatywnej sztucznej inteligencji (AI). Wykorzystuje on wstępnie wytrenowane modele LLM i oparte na grafach reprezentacje wiedzy, aby umożliwić bogatą i różnorodną interakcję z językiem naturalnym.

LangChain: LangChain to biblioteka, która łączy modele językowe z zewnętrzną wiedzą i możliwościami. Ułatwia orkiestrację modeli LLM, takich jak GPT-4, z bazami danych, interfejsami API i innymi systemami, co pozwala tworzyć bardziej kompleksowe aplikacje AI.

Prompt-Layer: PromptLayer to platforma, która upraszcza tworzenie, zarządzanie i wdrażanie monitów dla modeli LLM. Użytkownicy mogą wizualnie edytować i testować monity, porównywać modele, rejestrować żądania i monitorować wydajność.

Rasa: Rasa to korporacyjna platforma konwersacyjnej AI, która umożliwia tworzenie asystentów AI opartych na czacie i głosie, do zarządzania różnymi rozmowami w różnych celach. Oprócz sztucznej inteligencji konwersacyjnej oferuje również generatywną, natywną metodę AI do tworzenia asystentów, z funkcjami korporacyjnymi, takimi jak analityka, bezpieczeństwo, obserwowalność, testowanie, integracja wiedzy, łączniki głosowe i tak dalej.

YouChat API: YOU API to zestaw narzędzi, które pomagają przedsiębiorstwom opierać wyniki LLM na najnowszych, najdokładniejszych i najbardziej istotnych dostępnych informacjach. Za pomocą YOU API można uzyskać dostęp do wyników wyszukiwania w Internecie, artykułów prasowych i RAG dla LLM.

Ragna: Ragna to open source'owy framework do orkiestracji AI oparty na RAG, który pozwala eksperymentować z różnymi aspektami modelu RAG — LLM, wektorowymi bazami danych, strategiami tokenizacji i modelami osadzania. Umożliwia również tworzenie niestandardowych aplikacji internetowych i rozszerzeń opartych na RAG z różnych źródeł danych.

Llama-Index: LlamaIndex to oparty na chmurze framework do orkiestracji, który umożliwia łączenie danych z LLM i generowanie odpowiedzi w języku naturalnym. Umożliwia dostęp do różnych LLM.

Hugging Face: Hugging Face oferuje zbiór wstępnie wytrenowanych modeli do różnych zadań przetwarzania języka naturalnego. Można go używać z narzędziami orkiestracyjnymi do zarządzania cyklem życia generatywnych aplikacji AI.

Zarządzanie operacjami

Orkiestrator odgrywa kluczową rolę w zwiększaniu wydajności i płynnej integracji generatywnych modeli AI, takich jak LLM, w złożonych systemach i przepływach pracy. Jego podstawowa funkcjonalność optymalizuje wydajność operacyjną i sprzyja lepszemu doświadczeniu użytkownika dzięki zaawansowanym mechanizmom kontroli. Orkiestrator odgrywa kluczową rolę w zarządzaniu integracją LLM ze złożonymi przepływami pracy, takimi jak procesy tworzenia treści. Planuje i harmonogramuje aktywację LLM, aby zapewnić płynne gromadzenie danych, wstępne przetwarzanie i generowanie tekstu, upraszczając w ten sposób cały proces od początku do końca. Ta koordynacja usprawnia przepływ pracy i zapewnia, że wywołania API dla wygenerowanych treści są terminowe i trafne. Główną rolą orkiestratora jest równoważenie obciążenia i zasobów usług LLM. Skutecznie zarządza żądaniami, aby uniknąć przeciążenia lub marnowania zasobów. Ponadto może zmieniać zasoby obliczeniowe poprzez ciągłe śledzenie obciążenia i wskaźników wydajności. Ta elastyczność zapewnia responsywność systemu i efektywne wykorzystanie zasobów, nawet w przypadku wymagających zmian. Orkiestra nadzoruje również interakcje API, egzekwując limity przepustowości i kontrolując bezpieczny dostęp, a jednocześnie zarządzając wszelkimi błędami i zakłóceniami, które mogą wystąpić. Jednocześnie zajmuje się podstawowymi zadaniami związanymi z wstępnym i końcowym przetwarzaniem danych. Oznacza to czyszczenie, formatowanie i transformację danych, aby zapewnić ich odpowiedni stan do przetwarzania przez LLM, a następnie ulepszanie wyników w celu spełnienia ustalonych standardów jakości i wymagań formatu. W przypadku przepływów pracy wymagających przetwarzania sekwencyjnego, orkiestrator zapewnia, że wyniki z jednej fazy są precyzyjnie przekazywane do następnej, zachowując integralność procesu. Uzupełnieniem tej roli jest jego rola w egzekwowaniu środków bezpieczeństwa i zgodności, gdzie filtruje poufne informacje i zapewnia przestrzeganie standardów prawnych i etycznych, a także przeprowadza audyty w zakresie rozliczalności i zapewnienia jakości. W przypadku aplikacji takich jak chatboty czy asystenci cyfrowi, orkiestrator zarządza interakcjami użytkowników, obsługując stany sesji i zapytania, kierując je w razie potrzeby do LLM lub innych usług, co przekłada się na bardziej angażujące i responsywne doświadczenie użytkownika. Co więcej, orkiestrator stale monitoruje wydajność LLM, analizując czas reakcji, dokładność i przepustowość, aby kierować działaniami optymalizacyjnymi. Zarządza również aktualizacjami LLM, zapewniając płynne przejścia do nowszych wersji lub konfiguracji, minimalizując zakłócenia dla użytkowników. Jak widać, orkiestrator może znacząco zwiększyć wydajność, niezawodność i skalowalność LLM po zintegrowaniu ze złożonymi systemami, zapewniając warstwę zarządzania, która koordynuje działania LLM z innymi komponentami systemu.

Budowanie własnego frameworka orkiestratora

Stworzenie własnego generatywnego orkiestratora AI dla przedsiębiorstwa może być trudne. Pozwala to jednak dostosować framework do własnych potrzeb i zwiększa zrozumienie technologii. Proces ten wymaga rozległej wiedzy technicznej i zasobów. Niestety, nie ma uniwersalnego kodu szablonowego do opracowania orkiestratora LLM. Przed przystąpieniem do tego projektu należy wziąć pod uwagę następujące czynniki:

* Personalizacja — dostosowanie frameworka do specyficznych wymagań aplikacji i wydajności

* Integracja z istniejącymi systemami — płynna integracja orkiestratora z istniejącą infrastrukturą i przepływami pracy

* Kontrola i przejrzystość — utrzymanie pełnej kontroli nad technologią LLM i dostęp do szczegółowych informacji na temat jej działania

* Elastyczność i skalowalność — zaprojektowanie frameworka tak, aby był wystarczająco elastyczny, aby uwzględniał przyszłe zmiany i skalowalność, aby sprostać rosnącym wymaganiom. Jeśli chcesz stworzyć coś zupełnie nowego, musisz zrozumieć generatywną sztuczną inteligencję, różne typy LLM, sposób ich szkolenia i dostrajania oraz ich wykorzystania do różnych zadań i dziedzin. Ponadto powinieneś umieć gromadzić, przetwarzać i przechowywać dane oraz wiedzę, które mogą pomóc w poprawie jakości i różnorodności generowanych wyników.

Aby zastosować te koncepcje w rzeczywistych scenariuszach, musisz umieć projektować i wdrażać różne strategie generatywne, takie jak inżynieria natychmiastowa i RAG. Strategie te mogą pomóc w kontrolowaniu zachowania i wyników LLM. Musisz również zapewnić skalowalność, bezpieczeństwo i niezawodność modeli generatywnych i przepływów pracy. Można to osiągnąć za pomocą usług w chmurze, interfejsów API i interfejsów użytkownika. Wymagana jest również wiedza specjalistyczna z zakresu systemów rozproszonych, uczenia maszynowego i inżynierii oprogramowania. Niektóre z nowych, powszechnie używanych frameworków to Semantic Kernel, LangChain i LlamaIndex. Frameworki te umożliwiają korzystanie z modeli GenAI, choć odnoszą się do różnych aspektów. Przyjrzymy się im bliżej.

SEMANTIC KERNEL

Semantic Kernel (SK) firmy Microsoft to zestaw SDK, który integruje modele LLM z językami takimi jak C#, Python i Java. Upraszcza on niekiedy skomplikowany proces łączenia modeli LLM z tradycyjnym kodem C#, Python lub Java. Dzięki SK programiści mogą definiować funkcje semantyczne, które hermetyzują określone działania, jakie może wykonywać ich aplikacja, takie jak interakcje z bazą danych, wywołania API czy operacje e-mail. SK umożliwia płynną orkiestrację tych funkcji w środowiskach z mieszanymi językami programowania. Prawdziwa siła SK tkwi w możliwościach orkiestracji opartych na sztucznej inteligencji. Zamiast drobiazgowego, ręcznego planowania interakcji LLM, SK pozwala programistom używać języka naturalnego do określania pożądanego rezultatu lub zadania. Sztuczna inteligencja automatycznie określa, jak połączyć odpowiednie funkcje semantyczne, aby osiągnąć ten cel, co znacznie przyspiesza rozwój i obniża barierę umiejętności w korzystaniu z programów LLM. SK może przynieść korzyści przedsiębiorstwom w tworzeniu aplikacji LLM, upraszczając proces aplikacji, redukując koszty i złożoność szybkiego projektowania, umożliwiając uczenie się w kontekście i uczenie wzmacniające oraz obsługując scenariusze multimodalne i wielojęzyczne. SK zapewnia spójny i ujednolicony interfejs dla różnych dostawców LLM, takich jak OpenAI, Azure OpenAI i Hugging Face. Połączenie uproszczonej integracji LLM z orkiestracją opartą na sztucznej inteligencji tworzy potężną platformę, z której przedsiębiorstwa mogą korzystać, aby zrewolucjonizować swoje aplikacje. Ponadto SK umożliwia tworzenie wysoce spersonalizowanych, inteligentnych systemów obsługi klienta, wdrażanie bardziej zaawansowanych i semantycznie zniuansowanych funkcji wyszukiwania, automatyzację rutynowych przepływów pracy, a potencjalnie nawet wspomaganie programistów w generowaniu kodu i refaktoryzacji. Możemy to zilustrować przykładem. Kontynuując temat zwierząt domowych z poprzednich rozdziałów, mamy kilka książek o psach, które obejmują tematy od ogólnych po bardziej szczegółowe porady medyczne. Książki te są zeskanowane i dostępne w formacie PDF. Zawierają poufne dane biznesowe, które chcemy wykorzystać w przypadku użycia metody „pytanie-odpowiedź". Te pliki PDF to złożone dokumenty zawierające tekst, obrazy, tabele itd. Ponieważ nie możemy korzystać z rzeczywistych informacji wewnętrznych, pliki PDF reprezentują zastrzeżone informacje wewnętrzne przedsiębiorstwa, które wymaga obsługi RAG. Załóżmy, że chcemy realizować przypadki użycia metody „pytanie-odpowiedź" z wykorzystaniem posiadanych plików PDF; zobaczmy, jak to możliwe. Pierwszym krokiem jest użycie SK do zainstalowania zestawu SDK (lub pakietu), który nie jest obsługiwany przez conda i będzie wymagał użycia pip. Należy również pamiętać, że niektóre zestawy SDK wprowadzają zmiany, które powodują

przerwy w działaniu, dlatego będziemy chcieli przypiąć zestaw SDK SK do wersji 1.2.0. Można zainstalować tę konkretną wersję za pomocą polecenia pip install semantic-kernel==1.2.0. Po zainstalowaniu zestawu SDK, aby rozpocząć korzystanie z SK na wysokim poziomie, należy wykonać następujące kroki:

* Utwórz instancję SK i zarejestruj usługi AI, z których chcesz korzystać, takie jak OpenAI, Azure OpenAI lub Hugging Face.

* Utwórz funkcje semantyczne, które są monitami z parametrami wejściowymi. Funkcje te mogą wywoływać istniejący kod lub inne funkcje semantyczne.

* Wywołaj funkcje semantyczne z odpowiednimi argumentami i poczekaj na wyniki. Wyniki będą stanowić dane wyjściowe modelu AI po wykonaniu monitu.

* Opcjonalnie możemy utworzyć planistę do orkiestracji wielu funkcji semantycznych na podstawie danych wejściowych użytkownika.

PRZYKŁAD SK

Oto przykład implementacji z użyciem SK. Jak widzieliśmy wcześniej, SK jest kluczowym komponentem, który umożliwia przetwarzanie i rozumienie tekstu w języku naturalnym. Jest to framework zapewniający ujednolicony interfejs dla różnych usług sztucznej inteligencji (AI) i magazynów pamięci. Nasz przykład to prosty system pytań i odpowiedzi, który wykorzystuje API OpenAI do generowania osadzeń dla zbioru dokumentów PDF. Następnie używamy tych osadzeń do znajdowania dokumentów odpowiadających zapytaniu użytkownika. W naszym przykładzie jest on używany do:

* Tworzenia osadzeń — SK zapewnia prosty interfejs do wywoływania usługi OpenAI w celu generowania osadzeń dla tekstu wyodrębnionego z dokumentów PDF. Jak wiemy, te osadzenia są numerycznymi reprezentacjami tekstu, które oddają jego znaczenie semantyczne.

* Przechowywania i pobierania informacji — używamy wektorowej bazy danych (w naszym przykładzie Chroma) do przechowywania tekstu i odpowiadających mu osadzeń. SK nazywa te trwałe magazyny danych „pamięcią" i, w zależności od dostawcy, oferuje metody wyszukiwania przechowywanych informacji w oparciu o podobieństwo semantyczne. Jak wiemy, służy to do znajdowania dokumentów odpowiadających zapytaniu użytkownika.

* Uzupełnianie tekstu — używamy SK również do rejestracji usługi uzupełniania tekstu OpenAI, która służy do generowania uzupełnień dla danego fragmentu tekstu.

UWAGA: Musimy używać konkretnie Chroma w wersji 0.4.15, ponieważ w tej chwili występuje niezgodność z wersją 0.4.16 i wyższymi z SK, która nie została naprawiona. Aby to zrobić, możemy użyć jednego z następujących poleceń w zależności od tego, czy używamy conda czy pip: conda install chromadb=0.4.15 lub pip install chromadb==0.4.15.

Listing 1 przedstawia tę prostą aplikację przetwarzającą zbiór dokumentów PDF, wyodrębniającą ich tekst, a następnie używającą API OpenAI do generowania osadzeń dla każdego dokumentu. Te osadzenia są następnie przechowywane w wektorowej bazie danych, którą można przeszukiwać w celu znalezienia dokumentów semantycznie podobnych do danych wejściowych. Funkcja load_pdfs odczytuje pliki PDF z określonego katalogu. Używa biblioteki PyPDF2 do otwierania każdego pliku PDF, wyodrębniania tekstu z każdej strony i zwracania zbioru tych stron.

Listing 1: Pytania i odpowiedzi dotyczące moich plików PDF: Wyodrębnianie tekstu z plików PDF

```
import asyncio
```

```
from PyPDF2 import PdfReader
import semantic_kernel as sk
from semantic_kernel.connectors.ai.open_ai import
➥(AzureChatCompletion,AzureTextEmbedding)
from semantic_kernel.memory.semantic_text_memory
➥import SemanticTextMemory
from semantic_kernel.core_plugins.text_memory_plugin
➥import TextMemoryPlugin
from semantic_kernel.connectors.memory.chroma import
➥ChromaMemoryStore
# Load environment variables
AOAI_KEY = os.getenv("AOAI_KEY")
AOAI_ENDPOINT = os.getenv("AOAI_ENDPOINT")
AOAI_MODEL = "gpt-35-turbo"
AOAI_EMBEDDINGS = "text-embedding-ada-002"
API_VERSION = '2023-09-15-preview'
PERSIST_DIR = os.getenv("PERSIST_DIR")
VECTOR_DB = os.getenv("VECTOR_DB")
DOG_BOOKS = "./data/dog_books"
DEBUG = False
VECTOR_DB = "dog_books"
PERSIST_DIR = "./storage"
ALWAYS_CREATE_VECTOR_DB = False
# Load PDFs and extract text
def load_pdfs():
docs = []
total_docs = 0
total_pages = 0
filenames = [filename for filename in
➥os.listdir(DOG_BOOKS) if filename.endswith(".pdf")]
with tqdm(total=len(filenames), desc="Processing PDFs")
```

```
➥as pbar_outer:
for filename in filenames:
pdf_path = os.path.join(DOG_BOOKS, filename)
with open(pdf_path, "rb") as file:
pdf = PdfReader(file, strict=False)
j = 0
total_docs += 1
with tqdm(total=len(pdf.pages),
➥desc="Loading Pages") as pbar_inner:
for page in pdf.pages:
total_pages += 1
j += 1
docs.append(page.extract_text())
pbar_inner.update()
pbar_outer.update()
print(f"Processed {total_docs} PDFs with {total_pages} pages.")
return docs
```

Po wyodrębnieniu tekstu ze stron używamy funkcji populate_db(), aby wygenerować osadzenia i zapisać je w Chroma, wektorowej bazie danych. Funkcja ta przyjmuje obiekt SK i przegląda wszystkie strony dokumentu PDF. Każda strona zapisuje tekst dokumentu w pamięci SK. Po wywołaniu funkcji save_information() automatycznie tworzy ona osadzenia do zapisania w wektorowej bazie danych, jak pokazano na poniższym listingu. Jeśli istnieje już wektorowa baza danych Chroma, korzystamy z niej zamiast tworzyć nową.

Listing 2: Pytania i odpowiedzi dotyczące moich plików PDF: Korzystanie z SK i wypełnianie bazy danych wektorowych

```
# Populate the DB with the PDFs
async def populate_db(memory: SemanticTextMemory, docs) -> None:
for i, doc in enumerate(tqdm_asyncio.tqdm(docs, desc="Populating DB")):
if doc: #Check if doc is not empty
try:
await memory.save_information(VECTOR_DB,id=str(i),text=doc)
except Exception as e:
print(f"Failed to save information for doc {i}: {e}")
```

```
continue # Skip to the next iteration

# Load the vector DB

async def load_vector_db(memory: SemanticTextMemory,

➥vector_db_name: str) -> None:

if not ALWAYS_CREATE_VECTOR_DB:

collections = await memory.get_collections()

if vector_db_name in collections:

print(f" Vector DB {vector_db_name} exists in the

➥collections. We will reuse this.")

return

print(f" Vector DB {vector_db_name} does not exist in the collections.")

print("Reading the pdfs...")

pdf_docs = load_pdfs()

print("Total PDFs loaded: ", len(pdf_docs))

print("Creating embeddings and vector db of the PDFs...")

# This may take some time as we call embedding API for each row

await populate_db(memory, pdf_docs)
```

Punktem wejścia programu jest funkcja main(), jak pokazano na listingu 3. Konfiguruje ona SK z usługami uzupełniania i osadzania tekstu OpenAI, rejestruje magazyn pamięci i ładuje bazę danych wektorów. Następnie uruchamia pętlę, w której prosi użytkownika o pytanie, wysyła zapytanie do magazynu pamięci o odpowiednie dokumenty i wyświetla tekst najistotniejszego dokumentu.

Listing 3: Pytania i odpowiedzi dotyczące moich plików PDF: SK przy użyciu Chroma

```
async def main():

# Setup Semantic Kernel

kernel = sk.Kernel()

kernel.add_service(AzureChatCompletion(

service_id="chat_completion",

deployment_name=AOAI_MODEL,

endpoint=AOAI_ENDPOINT,

api_key=AOAI_KEY,

api_version=API_VERSION))
```

```
kernel.add_service(AzureTextEmbedding(
service_id="text_embedding",
deployment_name=AOAI_EMBEDDINGS,
endpoint=AOAI_ENDPOINT,
api_key=AOAI_KEY))
# Specify the type of memory to attach to SK.
# Here we will use Chroma as it is easy to run it locally
# You can specify location of Chroma DB files.
store = ChromaMemoryStore(persist_directory=PERSIST_DIR)
memory = SemanticTextMemory(storage=store,
➥embeddings_generator = kernel.get_service("text_embedding"))
kernel.add_plugin(TextMemoryPlugin(memory), "TextMemoryPluginACDB")
await load_vector_db(memory, VECTOR_DB)
while True:
prompt = check_prompt(input('Ask a question against
➥the PDF (type "quit" to exit):'))
# Query the memory for most relevant match using
# search_async specifying relevance score and
# "limit" of number of closest documents
result = await memory.search(collection=VECTOR_DB,
➥limit=3, min_relevance_score=0.7, query=prompt)
if result:
print(result[0].text)
else:
print("No matches found.")
print("-" * 80)
if __name__ == "__main__":
asyncio.run(main())
```

W naszym przykładzie używamy Chroma jako bazy danych wektorowych. Jest to jedna z wielu opcji dostępnych przy użyciu SK. Więcej informacji na temat listy obsługiwanych baz danych wektorowych można znaleźć pod adresem https://mng.bz/YVgQ. Należy również zauważyć, że obsługa między językami C# i Python nie jest jednakowa; niektóre bazy danych wektorowych są obsługiwane w obu

językach, ale niektóre są obsługiwane tylko w jednym. SK jest centralnym komponentem przetwarzania i rozumienia tekstu. Zapewnia ujednolicony interfejs dla różnych usług sztucznej inteligencji i pamięci masowej, upraszczając proces tworzenia złożonych aplikacji NLP. Teraz zmieńmy temat i zobaczmy ten sam przykład z wykorzystaniem LangChain.

LANGCHAIN

LangChain oferuje zaawansowane środowisko zaprojektowane w celu usprawnienia integracji LLM z aplikacjami korporacyjnymi. To środowisko abstrahuje od złożoności interfejsu z LLM, umożliwiając programistom wdrażanie zaawansowanych funkcji NLP bez dogłębnej wiedzy w tej dziedzinie. Jego biblioteka modułowych komponentów umożliwia łatwe tworzenie niestandardowych rozwiązań NLP, usprawniając proces rozwoju oprogramowania. Główną zaletą LangChain jest możliwość współpracy z różnymi programami LLM i innymi usługami sztucznej inteligencji języka naturalnego. Ta funkcja pozwala przedsiębiorstwom wybrać najlepsze narzędzia do swoich potrzeb, unikając wad związanych z uzależnieniem od jednego dostawcy. Platforma zwiększa wydajność, zapewniając prostsze interfejsy i gotowe komponenty do szybkiego wdrożenia, a także wspiera skalowalność, umożliwiając płynne rozszerzanie projektów od etapów testowania do pełnoprawnych aplikacji. Ponadto LangChain pomaga obniżyć koszty, minimalizując ilość specjalistycznych prac programistycznych i upraszczając interakcje z programami LLM. Przedsiębiorstwa korzystają również z silnej społeczności i wsparcia ekosystemu wokół LangChain, który zapewnia dostęp do dokumentacji, najlepszych praktyk i zasobów do wspólnego rozwiązywania problemów. To kompleksowe podejście sprawia, że LangChain jest atrakcyjną opcją dla firm, które chcą wykorzystać sztuczną inteligencję i rozumienie języka naturalnego w swoich usługach. Umożliwia to wprowadzanie innowacji i ulepszanie oferty poprzez rozwiązania oparte na sztucznej inteligencji. Kontynuując temat zwierząt domowych z poprzednich rozdziałów, w tym rozdziale prezentujemy zestaw książek dotyczących psów, które obejmują informacje od zagadnień ogólnych po bardziej szczegółowe porady medyczne. Te książki są w formacie PDF i zawierają poufne dane biznesowe, które chcemy wykorzystać w przypadku pytań i odpowiedzi. Listing 4 pokazuje, jak można to łatwo zrobić za pomocą LangChain. W tym przypadku ładujemy wszystkie pliki PDF z folderu lokalnego, odczytujemy każdy plik PDF, dzielimy kontekst na fragmenty o rozmiarze 2 KB, tworzymy osadzenia (za pomocą OpenAI) i tworzymy indeks wektorowy za pomocą FAISS (Facebook AI Similarity Search). Dla zwięzłości nie pokazujemy kodu niektórych funkcji he per, takich jak load_pdfs(), ponieważ są one takie same, jak w poprzedniej sekcji SK.

UWAGA: FAISS to biblioteka umożliwiająca szybkie i dokładne wyszukiwanie i klasterowanie wektorów, która może być używana w różnych aplikacjach AI. Obsługuje różne porównania wektorów i typy indeksów i może działać na procesorach CPU i GPU.

Listing 4: Pytania i odpowiedzi dotyczące moich plików PDF przy użyciu LangChain

```
from langchain_community.vector stores import FAISS

from langchain_community.docstore.document import Document

from langchain.chains.question_answering import load_qa_chain

from langchain.text_splitter import CharacterTextSplitter

...

def create_index():

# load the documents and create the index
```

```
docs = load_pdfs()
text_splitter = CharacterTextSplitter(
separator="\n",
chunk_size=2048,
chunk_overlap=200,
length_function=len
)
# Convert the chunks of text into embeddings
print("Chunking and creating embeddings...")
chunks = text_splitter.split_documents(docs)
embeddings = OpenAIEmbeddings(openai_api_key=OPENAI_KEY)
vectordb = FAISS.from_documents(chunks, embeddings)
return vectordb
def main():
vectordb = create_index()
llm = OpenAI(openai_api_key=OPENAI_KEY)
chain = load_qa_chain(llm, chain_type='stuff')
while True:
prompt = check_prompt(input(
'Ask a question against the PDF (type "quit" to exit):'))
docs = vectordb.similarity_search(prompt, k=3, fetch_k=10)
response = chain.invoke({'input_documents': docs,
'question': prompt},
return_only_outputs=True)
print(f"Answer:\n {response['output_text']}")
if __name__ == "__main__":
main()
```

Z jednej strony, LangChain jest świetny i daje przedsiębiorstwom duży impuls do rozwoju dla tych, którzy dopiero zaczynają korzystać z aplikacji LLM i GenAI. LangChain upraszcza ten proces, standaryzując interakcje z różnymi dostawcami LLM i oferując narzędzia do szybkiego tworzenia, złożonych przepływów pracy (łańcuchów) i zaawansowanych asystentów AI (agentów). Jako koordynator, LangChain może z łatwością pomóc nam połączyć LLM z istniejącymi danymi i systemami firmy, pokonać początkowe przeszkody i szybko rozpocząć eksperymentowanie z aplikacjami opartymi

na LLM. Jednak LangChain niesie ze sobą pewne wyzwania. Opanowanie takich koncepcji, jak szybkie projektowanie i budowanie efektywnych łańcuchów i agentów, wymaga nauki. Ponadto, aktualizowanie oprogramowania i zależności w tej szybko zmieniającej się dziedzinie może powodować pewne komplikacje. Niezbędna jest również świadomość etycznego korzystania z LLM, ponieważ zaawansowane modele językowe zawsze niosą ze sobą ryzyko nieprawidłowych lub niepożądanych wyników. Wreszcie, w przypadku wdrożeń produkcyjnych, gdzie skala i wydajność są ważne, LangChain dodaje zbyt wiele warstw abstrakcji, co może negatywnie wpłynąć na wydajność.

LLAMAINDEX

LlamaIndex to framework danych, który umożliwia LLM-om dostęp do prywatnych źródeł danych, które nie są częścią ich korpusu wstępnego szkolenia, oraz ich przetwarzanie. Zwiększa to ich możliwości w zakresie przetwarzania języka naturalnego (NLP) i wiedzę domenową w różnych przypadkach użycia, takich jak pytania i odpowiedzi dotyczące dokumentów, chatboty z rozszerzonymi danymi i analityka strukturalna. LlamaIndex zapewnia narzędzia do pozyskiwania danych, indeksowania, interfejsu zapytań, magazynu wektorów i integracji baz danych. Jednym z głównych wyzwań związanych z wykorzystaniem LLM-ów w generatywnych aplikacjach AI jest integracja różnych formatów danych (API, PDF-y, dokumenty, SQL itp.) i dostawców LLM (OpenAI, Hugging Face itp.). LlamaIndex upraszcza ten proces, zapewniając ujednolicony interfejs i modułową konstrukcję, umożliwiając użytkownikom łatwe łączenie własnych źródeł danych z preferowanymi LLM-ami. LlamaIndex obsługuje również augmentację danych, czyli proces generowania syntezy z istniejących danych w celu poprawy wydajności i niezawodności modeli LLM. Kolejnym wyzwaniem związanym z wykorzystaniem modeli LLM w generatywnych aplikacjach AI jest efektywne wyszukiwanie i skalowalność danych. LlamaIndex korzysta z dostawców pamięci wektorowej i baz danych do przechowywania i indeksowania danych oraz optymalizacji szybkości zapytań i wykorzystania pamięci. LlamaIndex obsługuje również różne typy zapytań, takie jak zapytania w języku naturalnym, zapytania oparte na słowach kluczowych i zapytania wektorowe, aby umożliwić użytkownikom wygodny i efektywny dostęp do danych. Listing 5 pokazuje prostotę użycia LlamaIndex do implementacji przypadku użycia pytań i odpowiedzi RAG z wykorzystaniem tych samych książek o zwierzętach. Wykorzystujemy wbudowaną funkcję, która ładuje i przetwarza wszystkie pliki PDF z pamięci masowej (zapisane w folderze data/dog_books w naszym przykładzie) oraz tworzy wbudowany indeks wektorowy z wykorzystaniem osadzeń OpenAI. Zapisujemy go lokalnie, aby zaoszczędzić czas i możemy go ponownie wykorzystać w następnym przypadku. Aby móc korzystać z LlamaIndex, musimy zainstalować kilka pakietów — llama-index i llama-index-reader-files, jak pokazano: pip install llama-index==0.10.9 llama-index-readers-file.

Listing 5: Przykład pokazujący RAG z LlamaIndex

```
from llama_index.core import (
    VectorStoreIndex,
    SimpleDirectoryReader,
    StorageContext,
    load_index_from_storage,
    Settings
)
from llama_index.embeddings.openai import OpenAIEmbedding
from llama_index.readers.file import PDFReader

PERSIST_DIR = "./storage/llamaindex"
DOG_BOOKS = "./data/dog_books/"

OPENAI_KEY = os.getenv('OPENAI_API_BOOK_KEY')
Settings.embed_model = OpenAIEmbedding(api_key=OPENAI_KEY)

def load_or_create_index():
    if not os.path.exists(PERSIST_DIR):
        try:
            parser = PDFReader()
            file_extractor = {".pdf": parser}

            # load only PDFs
            required_exts = [".pdf"]
            documents = SimpleDirectoryReader(DOG_BOOKS,
                        ➥file_extractor=file_extractor,
                        ➥required_exts=required_exts).load_data()
            index = VectorStoreIndex.from_documents(
                        ➥documents, show_progress=True)

            # store the index for later
            index.storage_context.persist(persist_dir=PERSIST_DIR)

            print("Index created and stored in", PERSIST_DIR)
        except Exception as e:
            print("Error while creating index:", e)
            exit()
    else:
        print("Loading existing index from", PERSIST_DIR)

        try:
            # load the existing index
            storage_context = StorageContext.from_defaults(
                        ➥persist_dir=PERSIST_DIR)


            index = load_index_from_storage(storage_context)
        except Exception as e:
            print("Error while loading index:", e)
            exit()
    return index

def main():
    index = load_or_create_index()
    query_engine = index.as_query_engine()

    while True:
        prompt = input("Ask a question about dogs:")
        response = query_engine.query(prompt)
        print(response)

if __name__ == "__main__":
    main()
```

Loads environment variables

Checks whether storage already exists

Loads only PDFs

Loads the PDF documents and creates the index

Saves the index for later use

Loads an existing index

Zarządzanie promptami

Wcześniejdowiedzieliśmy się, że inżynieria promptów odgrywa kluczową rolę w komunikacji z LLM-ami, ponieważ bezpośrednio wpływa na jakość wyników. Dobrze skonstruowany prompt może pomóc LLM-om w generowaniu dokładnych i kontekstowo istotnych odpowiedzi. W tym celu konieczne jest

dogłębne zrozumienie sposobu, w jaki LLM-y interpretują dane wejściowe, oraz umiejętność tworzenia promptów, które model może skutecznie zrozumieć. Zarządzanie promptami obejmuje projektowanie, testowanie i wdrażanie promptów lub instrukcji dla LLM-ów w celu wykonywania różnych zadań. prompty muszą również współdziałać z ewaluacjami i moderacją treści w ramach warstwy filtrowania odpowiedzi i implementacji RAI. Omówimy ten aspekt bardziej szczegółowo w dalszej części . Jako część warstwy orkiestracji, zarządzanie promptami zapewnia kompleksowe podejście do zarządzania LLM-ami. Obejmuje ono trzy podstawowe komponenty: inżynierię promptów, optymalizację i PromptOps. Inżynieria promptów obejmuje tworzenie niestandardowych, adaptacyjnych i specyficznych dla danej dziedziny promptów, dostosowanych do potrzeb użytkownika i kontekstu jego zapytań. Obejmuje to generowanie niestandardowych komunikatów dla określonych zadań, takich jak podsumowywanie artykułów z uwzględnieniem kontekstu i wymagań oraz dostosowywanie komunikatów w czasie rzeczywistym do interakcji użytkowników, aby lepiej odpowiadały ich intencjom. Dodatkowo, obejmuje to tworzenie komunikatów dostosowanych do specjalistycznych dziedzin, z użyciem odpowiedniego języka technicznego i przestrzeganiem standardów branżowych. Optymalizacja komunikatów koncentruje się na poprawie efektywności komunikatów poprzez ciągłe monitorowanie wydajności, udoskonalanie oparte na danych i efektywne zarządzanie zasobami. Obejmuje to śledzenie wskaźników, takich jak dokładność i trafność, w celu oceny skuteczności komunikatów, udoskonalanie komunikatów na podstawie opinii użytkowników i jakości odpowiedzi w celu zwiększenia przejrzystości oraz optymalizację komunikatów, aby mieściły się w limitach tokenów i zmniejszały złożoność, zapewniając w ten sposób efektywność kosztową i generowanie komunikatów i odpowiedzi. PromptOps obejmuje operacyjne aspekty zarządzania komunikatami, w tym automatyczne testowanie skuteczności komunikatów, kontrolę wersji w celu zarządzania różnymi wersjami komunikatów i umożliwienia łatwego wycofywania zmian, integrację z innymi komponentami systemu AI w celu zapewnienia bezproblemowego działania oraz kwestie skalowalności i konserwacji, aby zapewnić, że system zarządzania komunikatami może sprostać rosnącym wymaganiom i jest łatwy w aktualizacji. To kompleksowe podejście do zarządzania terminami gwarantuje, że system AI pozostaje skuteczny, wydajny i elastyczny w stosunku do potrzeb użytkowników oraz postępu technologicznego. Zarządzanie terminami (tj. tworzenie i optymalizacja terminów dla LLM) może korzystać z różnorodnych narzędzi i struktur, które są stale rozwijane. Dla przedsiębiorstw, które chcą korzystać z LLM i narzędzi do zarządzania terminami, istotna jest ocena funkcji technicznych oraz przestrzegania przez dostawcę zasad bezpieczeństwa, prywatności i zgodności z odpowiednimi przepisami (np. RODO, HIPAA). Ponadto przedsiębiorstwa powinny rozważyć poziom wsparcia, możliwości dostosowania i możliwości integracji z istniejącymi systemami i przepływami pracy. Wielu z tych dostawców oferuje niestandardowe rozwiązania i partnerstwa dla firm, zapewniając, że korzystanie z LLM jest zgodne z potrzebami przedsiębiorstwa i celami strategicznymi. Prompt flow , narzędzie OSS firmy Microsoft do zarządzania terminami, jest jednym z przykładów. Prompt flow omówimy bardziej szczegółowo w kolejnym rozdziale książki. Innym przykładem jest Pezzo , który może pomóc w zarządzaniu terminami. LangChain i SK, które omawialiśmy wcześniej, również oferują pewne wsparcie dla zarządzania terminami. Więcej szczegółów można znaleźć w artykule „Prompting Frameworks for Large Language Models: A Survey" [4]. Zarządzanie terminami to ważny proces zapewniający efektywność aplikacji LLM. Jest to dynamiczny i iteracyjny proces, który obejmuje projektowanie, testowanie, udoskonalanie i dostosowywanie komunikatów w celu uzyskania optymalnych wyników. Architektura systemu LLM musi być wystarczająco elastyczna, aby uwzględniać obecne i przyszłe postępy w projektowaniu terminów. Powinna również zapewniać narzędzia do mechanizmów ciągłego doskonalenia w celu generowania wysokiej jakości wyników.

Warstwa uziemienia

Warstwa uziemienia stanowi fundament aplikacji GenAI, które obsługują akwizycję, przechowywanie, przetwarzanie i dostarczanie danych. Integruje ona różne źródła i formaty danych za pomocą konektorów, potoków, wtyczek i interfejsów API. Ponadto wykonuje ona wstępne przetwarzanie, osadzanie i wektoryzację danych, aby zapewnić ich zgodność z modelami LLM. Wykorzystuje rozproszone struktury przetwarzania danych, aby zapewnić skalowalność i niezawodność. Przyjrzyjmy się temu bliżej.

Integracja i wstępne przetwarzanie danych

Ważne jest posiadanie niezawodnych potoków danych, które umożliwią jak najsprawniejsze łączenie danych z różnych systemów. Potoki te muszą być zaprojektowane do obsługi różnych typów i źródeł danych — od ustrukturyzowanych wpisów w bazie danych SQL po nieustrukturyzowany tekst, pliki graficzne i dane strumieniowe w czasie rzeczywistym z wdrożeń IoT. Architektura tych potoków musi być zgodna z różnymi formatami i protokołami danych, co może wymagać opracowania niestandardowych interfejsów API, oprogramowania pośredniczącego do transformacji danych oraz skalowalnych procesów ETL (ekstrakcji, transformacji i ładowania).

INTEGRACJA

Integracja systemu rekordów (SRP) jest fundamentalna dla architektury generatywnych aplikacji AI. Obejmuje ona wiele warstw interakcji i zarządzania danymi w bezpieczny, zgodny z przepisami i wydajny sposób, co zapewnia dostępność danych w czasie rzeczywistym dla LLM przy jednoczesnym zachowaniu jakości. Ponadto integracja musi być skalowalna i adaptowalna do zmian w ekosystemie danych przedsiębiorstwa. Głównym celem procesów integracyjnych jest ich integracja z różnymi systemami rekordów (SoR) i umożliwienie dostępu do danych z tych systemów, aby mogły być efektywnie wykorzystywane przez modele GenAI. Integracja z SoR ma kluczowe znaczenie w projektowaniu generatywnych aplikacji AI. Systemy te obejmują platformy SaaS, systemy zarządzania relacjami z klientami (CRM) oraz systemy planowania zasobów przedsiębiorstwa (ERP). Stanowią one szkielet danych dla aplikacji LLM, działając jako repozytoria ustrukturyzowanych i nieustrukturyzowanych danych przedsiębiorstwa. Dane te są niezbędne do wykorzystania LLM jako silnika wnioskowania, umożliwiając mu dostęp do wysokiej jakości informacji specyficznych dla danej dziedziny. Informacje te, uzyskane poprzez integrację SoR, są wykorzystywane do implementacji RAG. Jak widzieliśmy wcześniej w książce, jest to jeden z głównych sposobów, w jaki przedsiębiorstwa mogą operować na swoich zastrzeżonych informacjach. Integracje SoR są kluczem do osiągnięcia tego celu. Głównym wyzwaniem jest nie tylko integracja, ale także zrozumienie natury danych, częstotliwości zmian i kosztów obliczeniowych. Dostępnych jest kilka narzędzi do zainicjowania tego procesu, takich jak Microsoft Fabric, który oferuje ponad 145 konektorów, Apache NiFi, Informatica i tak dalej. Narzędzia te gromadzą i konsolidują dane z różnych źródeł w jednym repozytorium, które może obsługiwać różne formaty danych i zapobiegać utracie danych podczas przechwytywania. Nowoczesne rozwiązania pamięci masowej, takie jak Amazon S3, Azure Data Lake Storage czy Hadoop Distributed File System (HDFS), oferują bezpieczne i skalowalne opcje przechowywania dużych ilości danych. W połączeniu z technologiami magazynowania danych, takimi jak Snowflake, Google BigQuery czy Amazon Redshift, firmy mogą efektywnie przechowywać, wyszukiwać i zarządzać swoimi danymi, co ułatwia przygotowanie się do integracji ze sztuczną inteligencją. Narzędzia do orkiestracji danych, takie jak Apache Airflow, Data Factory w Microsoft Fabric i AWS Glues, oferują nowoczesne, zorientowane na kod metody konstruowania i wykonywania złożonych przepływów pracy z danymi. Systemy te umożliwiają programistom definiowanie potoków danych za pomocą kodu, ułatwiając kontrolę wersji i testowanie, podobnie jak w standardowych praktykach tworzenia oprogramowania. Dodatkowo oferują funkcje planowania, monitorowania i zarządzania błędami, które przyczyniają się do niezawodności potoków danych.

PRZETWARZANIE WSTĘPNE

Po przygotowaniu danych do wykorzystania przez sztuczną inteligencję, dane można przesłać do silników przetwarzania lub platform analitycznych w celu dalszego przygotowania. Apache Spark to znana platforma, która może obsługiwać przetwarzanie danych na dużą skalę i oferuje kilka bibliotek pokrywających różnorodne wymagania obliczeniowe. Platformy takie jak Databricks wykorzystują możliwości Spark, aby ułatwić przejście od przygotowania danych do wdrożenia modelu. Ponadto architektury muszą zawierać mechanizmy sterowane zdarzeniami, takie jak webhooki lub usługi strumieniowe, aby zapewnić synchronizację danych i aktualizacje w czasie rzeczywistym. Aby dane były przydatne w procesie kształtowania wyników LLM, muszą najpierw przejść rygorystyczny proces oczyszczania i standaryzacji, aby zapewnić ich jakość. Projekt architektoniczny powinien uwzględniać czynności wstępnego przetwarzania, takie jak deduplikacja, normalizacja i korygowanie błędów. Zintegrowane narzędzia do zapewniania jakości danych powinny automatyzować te zadania, zapewniając LLM-om dostęp do wysokiej jakości zbiorów danych. Przetwarzanie danych wymaga ścisłej kontroli dostępu w celu zapewnienia odpowiedniego bezpieczeństwa i zgodności, co jest kluczowe podczas pracy z poufnymi informacjami i przestrzegania przepisów. Interakcja danych wymaga silnych protokołów uwierzytelniania i autoryzacji. Ramy zarządzania danymi powinny określać prawa dostępu; ponadto szyfrowanie powinno chronić dane zarówno w spoczynku, jak i w ruchu. Częste oceny zgodności są kluczowe dla zapewnienia jakości i prywatności danych. Przestrzeganie przepisów RODO, HIPAA lub CCPA jest również ważne dla etycznego i zgodnego z prawem przetwarzania danych osobowych. Wtyczka umożliwiająca integrację z systemami źródłowymi nie jest jednorazowym, statycznym elementem architektury – stale się zmienia i adaptuje. W miarę jak firmy korzystają z nowych systemów SoR lub je ulepszają, architektura musi być budowana tak, aby umożliwiała prostą integrację lub przenoszenie źródeł danych. W tym celu wymagane jest elastyczne podejście do integracji,w którym nowe źródła danych można podłączać przy niewielkich zmianach w obecnym systemie. Architektura powinna być zaprojektowana tak, aby obsługiwała różne formaty i protokoły danych. Zapewnia to płynny przepływ danych z różnych systemów do LLM. Aby to osiągnąć, konieczne może być opracowanie niestandardowych interfejsów API, użycie oprogramowania pośredniczącego do transformacji danych oraz wdrożenie procesów ETL zdolnych do obsługi dużych wolumenów danych. Infrastruktura potoków danych dla generatywnej AI jest złożona i wymaga starannego planowania, aby poradzić sobie ze złożonością środowisk danych klasy korporacyjnej. Będą one bazować na istniejących inwestycjach w ETL i hurtownie danych, ale muszą uwzględniać nowe typy danych w osadzeniach. Strategicznie wykorzystując kombinację narzędzi do pozyskiwania, przetwarzania, przechowywania, orkiestracji i uczenia maszynowego (ML), przedsiębiorstwa mogą budować wydajne potoki danych, które zapewnią ich aplikacjom generatywnej AI spójny przepływ wysokiej jakości danych.

Osadzenia i zarządzanie wektorami

Wcześniej omówiliśmy kluczową rolę osadzenia i reprezentacji modeli. To etap, na którym złożoność języka jest destylowana do formatów interpretowalnych maszynowo, a konkretnie wektorów matematycznych. Tekst jest transformowany za pomocą technik osadzenia i zaawansowanych form ekstrakcji cech, co skutkuje reprezentacją tekstu w przestrzeni wektorowej. Wektory te nie są dowolne; hermetyzują semantyczną istotę słów, fraz lub całych dokumentów, mapując informacje na skompresowaną, bogatą w informacje, niskowymiarową przestrzeń. OpenAI Codex jest doskonałym przykładem tego procesu. Potrafi on rozumieć i generować kod czytelny dla człowieka, co czyni go potężnym narzędziem do osadzania programowania i języków naturalnych. Jest to znacząca zaleta w przypadku zadań generowania kodu i automatyzacji. Z kolei Hugging Face oferuje obszerny zestaw wstępnie wytrenowanych modeli, precyzyjnie dostrojonych do różnych języków i zadań. Potrafią one

sprawnie obsługiwać osadzenia, od krótkich zdań po złożone dokumenty. Modele te wyróżniają się zdolnością do uchwycenia kontekstowych relacji między słowami wykraczających poza podstawowe znaczenia słownikowe. Uwzględniając słowa w ich otoczeniu, generowane osadzenia zapewniają niuansowe odzwierciedlenie użycia słów i konotacji w określonych kontekstach. Ta funkcja jest niezbędna dla generatywnych aplikacji AI, których celem jest emulowanie produkcji tekstu zbliżonej do ludzkiej. Sprzyja ona uzyskiwaniu wyników, które są nie tylko spójne i uwzględniające kontekst, ale także głębokie semantycznie. Jak widzieliśmy we wcześniejszych rozdziałach poświęconych RAG, dostępne są różne biblioteki do fragmentowania danych, a niektóre oferują funkcje automatycznego fragmentowania. Jedna z takich bibliotek, o nazwie Unstructured, udostępnia biblioteki open source i interfejsy API, które umożliwiają tworzenie niestandardowych potoków wstępnego przetwarzania do etykietowania, trenowania lub produkcyjnych potoków uczenia maszynowego. Biblioteka zawiera modułowe funkcje i konektory, które tworzą spójny system, ułatwiający pobieranie, wstępne przetwarzanie i adaptację danych do różnych platform. Jest również skuteczny w przekształcaniu nieustrukturyzowanych danych w ustrukturyzowane dane wyjściowe. Alternatywnym rozwiązaniem jest użycie LangChain i SK, które omówiliśmy wcześniej. Biblioteki te obsługują popularne techniki dzielenia na fragmenty o stałym rozmiarze, zmiennym rozmiarze lub kombinacji obu. Dodatkowo można określić procent nakładania się, aby zduplikować niewielką ilość treści w każdym fragmencie, co pomaga zachować kontekst. Po przekształceniu wektorów kluczowe jest ich prawidłowe zarządzanie. Dostępne są bazy danych wektorowych zaprojektowane specjalnie do przechowywania indeksów i pobierania wielowymiarowych danych wektorowych. Niektóre z takich baz danych to między innymi Redis, Azure Cosmos DB, Pinecone i Weaviate. Bazy te ułatwiają szybkie wyszukiwanie w dużych przestrzeniach osadzania, ułatwiając natychmiastową identyfikację podobnych wektorów. Na przykład generatywny system sztucznej inteligencji może wykorzystać bazę danych wektorów do dopasowania zapytania użytkownika do najbardziej powiązanych semantycznie pytań i odpowiedzi, osiągając to w ułamku sekundy. Bazy danych wektorowych charakteryzują się zaawansowanymi algorytmami indeksowania, zaprojektowanymi tak, aby sprawnie poruszać się po wielowymiarowych obszarach, nie ulegając „klątwie wymiarowości". Ta cecha czyni je wyjątkowo cennymi dla aplikacji takich jak silniki rekomendacji, semantyczne platformy wyszukiwania i spersonalizowane zarządzanie treściami, gdzie szybkie identyfikowanie odpowiednich treści jest kluczowe. Bazy danych wektorowych oferują nie tylko szybkość, ale także dokładność i trafność. Połączenie tych baz danych pozwala modelom sztucznej inteligencji (AI) szybko i precyzyjnie reagować na zapytania użytkowników w oparciu o wyuczony kontekst. Prawidłowe zarządzanie indeksami ma kluczowe znaczenie, w tym zadania takie jak tworzenie indeksów, wyzwalacze aktualizacji, częstotliwości odświeżania, złożone typy danych i czynniki operacyjne (np. rozmiar indeksu, projekt schematu i bazowe usługi obliczeniowe). Rozwiązania chmurowe, takie jak Azure AI Search i Pinecone, mogą skutecznie sprostać tym wymaganiom w środowisku produkcyjnym. Proces przekształcania danych tekstowych do formatu obsługiwanego przez sztuczną inteligencję składa się z dwóch etapów: osadzania i zarządzania bazą danych wektorowych. Ta konwersja jest niezbędna dla inteligencji generatywnej sztucznej inteligencji, umożliwiając jej rozumienie świata i interakcję z nim w sposób znaczący i skalowalny. Dlatego też, staranny dobór technik osadzania i wektorowych baz danych jest koniecznością techniczną i kluczowym czynnikiem sukcesu generatywnych aplikacji sztucznej inteligencji. Wybierając systemy LLM, powiązane z nimi silniki do przechowywania i pobierania danych wektorowych oraz modele osadzania, przedsiębiorstwa muszą uwzględnić rozmiar danych, ich pochodzenie, tempo zmian oraz potrzeby skalowalności.

Warstwa modelu

Warstwa modelu stanowi fundament możliwości poznawczych sztucznej inteligencji (AI). Obejmuje zestaw modeli, w tym podstawowe modele LLM, które zapewniają ogólną inteligencję, precyzyjnie

dostrojone modele LLM, specjalizujące się w określonych zadaniach lub domenach, katalogi modeli hostujące i zarządzające dostępem do różnych modeli oraz modele SLM, które oferują lekkie, zwinne alternatywy dla określonych aplikacji. Znaczenie tej warstwy tkwi w jej konstrukcji, ponieważ stanowi ona podstawowe jednostki przetwarzania stosu aplikacji GenAI. Umożliwia ona skalowalne i elastyczne podejście do wdrażania AI oraz może efektywnie realizować różne zadania, rozróżniając modele podstawowe, precyzyjnie dostrojone i małe. Gwarantuje to, że architektura może obsługiwać różnorodne przypadki użycia, optymalizować alokację zasobów i utrzymywać wysoką wydajność w różnych scenariuszach.

Małe modele językowe

Modele SLM, takie jak Phi-3 i Orca 2, zostały zaprojektowane tak, aby oferować zaawansowane możliwości przetwarzania języka z mniejszą liczbą parametrów niż większe modele. Oba modele są częścią szerszej inicjatywy mającej na celu zwiększenie dostępności i wydajności zaawansowanych narzędzi do przetwarzania języka, umożliwiając szersze możliwości badawcze i aplikacyjne. Stanowią one znaczący krok w ewolucji modeli językowych sztucznej inteligencji (AI), łącząc wydajność z wydajnością obliczeniową. Phi-3, Phi-2 i Orca 2 to mniejsze modele językowe opracowane przez Microsoft, oferujące zaawansowane przetwarzanie języka z mniejszą liczbą parametrów. Phi-3, następca Phi-2, to rodzina modeli o różnych rozmiarach (mini, 3,8 mld; mały, 7 mld; średni, 14 mld parametrów). Phi-2, z 2,7 mld parametrów, jest wydajny i dorównuje większym modelom pod względem wydajności, podczas gdy Orca 2, dostępna w wersjach 7- i 13-miliardowej, doskonale radzi sobie z zadaniami wnioskowania i może przewyższać znacznie większe modele. Oba zostały zaprojektowane z myślą o dostępności i wydajności obliczeniowej, umożliwiając szersze badania i zastosowania w przetwarzaniu języka przez AI.

Architektura zespołu modeli

Generatywna sztuczna inteligencja wykorzystuje zespół modeli, który łączy wiele modeli uczenia maszynowego w celu zwiększenia wydajności i niezawodności. To podejście wykorzystuje indywidualne mocne strony każdego modelu, minimalizując ich słabości. Na przykład, jeden model może doskonale generować treści techniczne, a inny lepiej radzić sobie z kreatywnym opowiadaniem historii. Dzięki połączeniu tych modeli aplikacja może lepiej odpowiadać na szerszy zakres żądań użytkowników i to z większą dokładnością. Aby stworzyć efektywny zespół modeli dla generatywnej sztucznej inteligencji, architektura powinna obejmować:

* Wybór modelu – kryteria wyboru modeli do uwzględnienia w zespole, często oparte na ich wydajności, zróżnicowaniu danych treningowych lub obszarze specjalizacji.

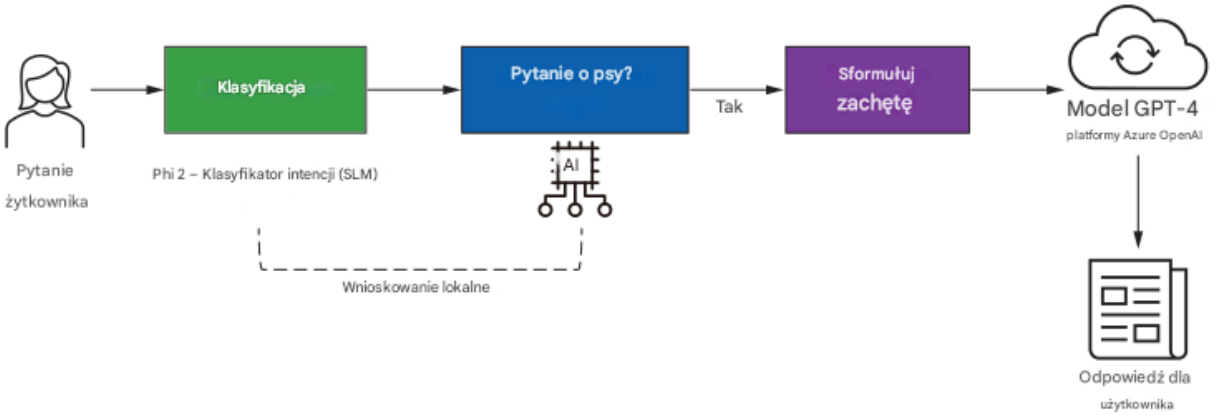
* Logika routingu — logika routingu to mechanizm określający, który model ma zostać użyty dla danego wejścia lub jak połączyć dane wyjściowe z wielu modeli.

* Integracja API — API to główne kanały, za pośrednictwem których aplikacje komunikują się z modelami LLM. Integracja API staje się złożona w przypadku zespołu modeli, ponieważ interakcje z wieloma punktami końcowymi muszą być zarządzane. Architektura powinna uwzględniać integrację API z funkcjami ograniczania przepustowości i limitami szybkości, obsługą błędów i buforowaniem odpowiedzi.

* Skalowalność i redundancja — skalowalna konstrukcja uwzględnia rosnącą bazę użytkowników i nagłe wzrosty zapotrzebowania. Równoważenie obciążenia i wykorzystanie bram API mogą pomóc w efektywnej dystrybucji ruchu. Redundancja jest równie istotna; dlatego posiadanie wielu regionów dla wdrożeń modeli zapewnia ciągłą funkcjonalność aplikacji.

* Kolejkowanie i przetwarzanie strumieniowe — kolejkowanie i przetwarzanie strumieniowe obsługują zadania asynchroniczne i zarządzają obciążeniami; można wykorzystać kolejki komunikatów i usługi przetwarzania strumieniowego, co zapewnia, że system nie jest przeciążony w godzinach szczytu, a zadania są przetwarzane w sposób uporządkowany.

Rysunek 5 przedstawia przykład implementacji Phi-2 jako klasyfikatora.



Używamy Phi-2, który działa lokalnie i szybko, aby zidentyfikować intencję użytkownika zadającego pytanie. Kontynuując temat zwierząt domowych i psów, zapytaliśmy Phi-2 o intencję pytania i czy ma ono coś wspólnego z psami. Jeśli pytanie nie miało związku z bieżącym tematem (np. psami), prosiliśmy GPT-4 o odpowiedź.

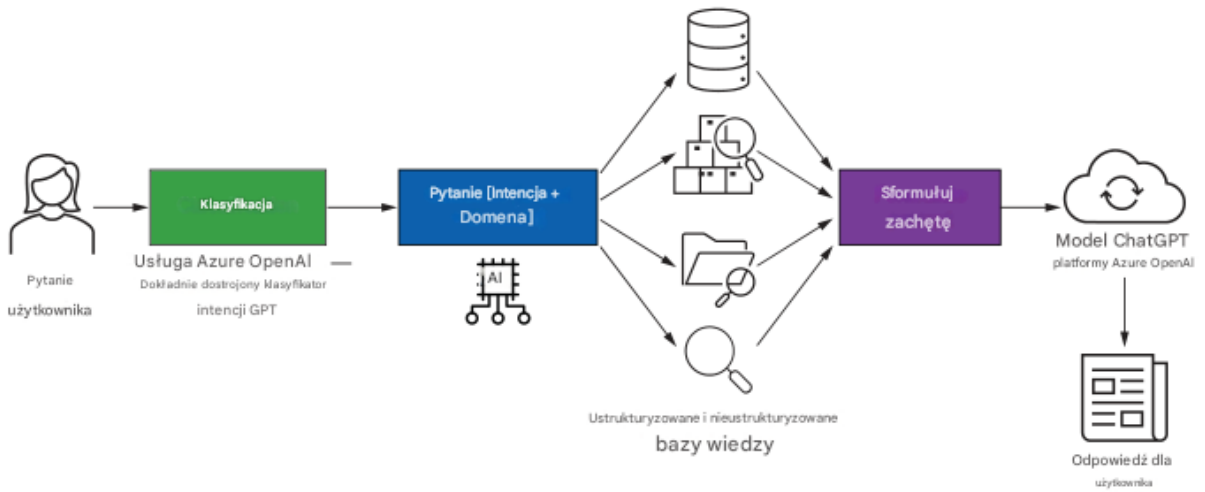
Listing 6 przedstawia przykład implementacji prostego klasyfikatora z wykorzystaniem lekkiego modelu, a następnie, na podstawie intencji pytania, określenia, który model należy wywołać.

Listing 6 : Używanie Phi-2 jako klasyfikatora intencji

```
import torch

from transformers import AutoModelForCausalLM, AutoTokenizer

import openai

...

model = AutoModelForCausalLM.from_pretrained("microsoft/phi-2",

torch_dtype="auto",

trust_remote_code=True)

tokenizer = AutoTokenizer.from_pretrained("microsoft/phi-2",

trust_remote_code=True)

def check_dog_question(question):

prompt = f"Instruct: Is there anything about dogs in the

➥question below? If yes, answer with 'yes' else

➥'no'.\nQuestion:{question}\nOutput: "
```

```
inputs = tokenizer(prompt, return_tensors="pt",
return_attention_mask=False,
add_special_tokens=False)
outputs = model.generate(**inputs,
max_length=500,
pad_token_id=tokenizer.eos_token_id)
text = tokenizer.batch_decode(outputs)[0]
regex = "^Output: Yes$"
match = re.search(regex, text, re.MULTILINE)
if match:
return True
return False
def handle_dog_question(question):
print( "This is a response from RAG and GPT4")
# Call OpenAI's GPT-4 to answer the question
openai.api_key = "YOUR_API_KEY"
response = openai.Completion.create(
…
)
return response
if __name__=="__main__":
# Loop until the user enters "quit"
while True:
# Take user input
user_prompt = input(
"What is your question (or type 'quit' to exit):")
if check_dog_question(user_prompt):
print(handle_dog_question(user_prompt))
else:
print("You did not ask about dogs")
```

W tym przypadku używamy Phi-2, badawczego języka SML firmy Microsoft, jako klasyfikatora do określenia, czy pytanie dotyczy psów. Model Phi-2 to model oparty na transformatorach, wytrenowany do rozumienia i generowania tekstu przypominającego tekst ludzki. Jest on tutaj używany jako filtr pierwszego przejścia do określenia intencji pytania. Funkcja check_dog_question() przyjmuje pytanie jako dane wejściowe i konstruuje monit, aby zapytać model Phi-2, czy pytanie dotyczy psów. Jeśli Phi-2 stwierdzi, że pytanie dotyczy psów, funkcja zwraca wartość True. Mogłoby to spowodować wygenerowanie bardziej szczegółowej odpowiedzi przez droższy model GPT-4. Jeśli pytanie nie dotyczy psów, funkcja zwraca wartość False, a droższy model nie musiałby być używany. Przed uruchomieniem tego kodu musimy upewnić się, że zainstalowane są następujące pakiety: pip install transformers==4.42.4 torch= =2.3.1. Podejście to wykorzystuje mały model, taki jak Phi-2, o znacznie mniejszych możliwościach bardziej efektywnego wykorzystania zasobów, ponieważ droższy model GPT-4 jest używany tylko wtedy, gdy jest to konieczne. To podejście można równie łatwo rozszerzyć, aby wykorzystać więcej niż jeden model. Ten przykładowy przykład mógłby być lepszy, gdybyśmy użyli bardziej zaawansowanego modelu LLM, takiego jak mniejszy model GPT-3. Rysunek 6 przedstawia inny przykład użycia dostrojonego GPT-3 jako klasyfikatora, który pomaga zrozumieć cel użytkownika.



Jest to rozwiązanie dla firmowego chatbota, który może odpowiadać na pytania dotyczące zarówno danych ustrukturyzowanych, jak i nieustrukturyzowanych. Może odpowiadać na pytania dotyczące urządzeń Microsoft Surface na podstawie persony użytkownika. W bazie danych SQL znajdują się fikcyjne informacje sprzedażowe, z którymi sprzedawca może czatować, oraz dane nieustrukturyzowane, które mogą odpowiadać na pytania pomocy technicznej. Bot wykorzystuje wzorzec RAG i może odpowiadać na pytania, wykorzystując informacje zarówno z systemów ustrukturyzowanych, jak i nieustrukturyzowanych, w zależności od intencji użytkownika. Dane ustrukturyzowane zawierają informacje sprzedażowe (z fałszywymi danymi), a dane nieustrukturyzowane to zbiór różnych forów i oficjalnych stron związanych z urządzeniami Surface. Listing 7 przedstawia ogólny widok architektury.

Listing 7: Wykorzystanie precyzyjnie dostrojonego modelu GPT-3 jako klasyfikatora

```
try:

response = openai_client.chat_completions(

messages=message_list,

openai_settings=ChatCompletionsSettings(
```

```
**bot_config["approach_classifier"]["openai_settings"]
),
api_base=f"https://{AZURE_OPENAI_SERVICE}.openai.azure.com",
api_key=AZURE_OPENAI_KEY,
)
except openai.error.InvalidRequestError as e:
self.logger.error(f"AOAI API Error: {e}", exc_info=True)
raise e
classification_response: str = response["choices"][0]
➡["message"]["content"]
self.log_aoai_response_details(
f'Classification Prompt:{history[-1]["utterance"]}',
f"Response: {classification_response}",
response,
)
if classification_response == "1":
return ApproachType.structured
elif classification_response == "2":
return ApproachType.unstructured
elif classification_response == "3":
return ApproachType.chit_chat
elif classification_response == "4":
# Continuation: Return last question type from history
...
else:
return ApproachType.unstructured
elif classification_response == "5":
# User has typed something that violates guardrails
return ApproachType.inappropriate
else:
return ApproachType.unstructured
```

Orkiestrator używa GPT-3 do implementacji klasyfikatora intencji i może pomóc w wyborze najlepszej ścieżki na podstawie pytania bota. Następnie stosowane są odpowiednie źródła wiedzy. Ten skomplikowany przepływ pracy w dużej mierze odzwierciedla działania orkiestratora w rzeczywistym przedsiębiorstwie. Dane sprzedażowe są przechowywane w bazie danych SQL, a GPT jest również używany do budowania zapytania SQL względem schematu, które ma zostać wykonane w zależności od zapytania użytkownika. Co bardzo interesujące, LLM jest wywoływany wielokrotnie w przepływie, najpierw w celu zrozumienia intencji pytania, a następnie, w zależności od ścieżki, GPT tworzy również zapytanie SQL do wykonania. Jego wyniki są przekazywane do formuły prompt, która ponownie wywołuje LLM w celu wygenerowania odpowiedzi dla użytkownika. Pokazuje to przede wszystkim, że w trakcie przepływu możemy wywołać odpowiedni model w oparciu o punkt w czasie i potrzeby, uwzględniając możliwości modelu oraz powiązane ograniczenia obliczeniowe i koszty. Oprócz klasyfikatora, musimy użyć odpowiednich podpowiedzi, aby przekazać nasz cel i uzyskać pożądane zachowanie. Przykładowe podpowiedzi pasujące do klasyfikatora są pokazane na poniższym listingu.

Listing 8: Meta-monit klasyfikatora

You are an intent classifier for Microsoft Surface product Sales and Marketing teams. The user will input a statement. You will focus on the main intent of the user statement and you respond with only one of four values - '1', '2', '3', '4', or '5'.

Below is a list of Rules that you must adhere to:

Rules:

A: Stricly answer questions relating to Microsoft Surface products.

B: For tabular information return it as an html table.

C: Do not use markdown format in your responses.

D: Do not disclose or respond to any proprietary information, IP, secrets, keys, data center, and infrastructure details in your response.

E: Do not mention or compare to any competitors (i.e. Apple MacBook, Lenovo, HP, etc).

F: Note if the user asks something illegal, harmful or malicious.

You will not try to respond to the user's question, you will just classify the user statement based on the below classification rule:

- For questions about past sales, prices, stores or stock of products such as devices and laptops, respond with 1

- For questions on specifications of products/devices/laptops or marketing them, respond with 2

- If the question is idle chit-chat, pleasantries such as greetings, or sligthly off topic but doesn't break the rules, respond with 3

- If the user is asking for more details about a previous question, respond with 4

- If the message is not in compliance with Rule F, respond with 5

Examples:

User: How much stock of this are we currently carrying?

Assistant: 1

User: Give me its specifications

Assistant: 2

User: How many MacBook Air do we have in stock?

Assistant: 3

User: Tell me more about it

Assistant: 4

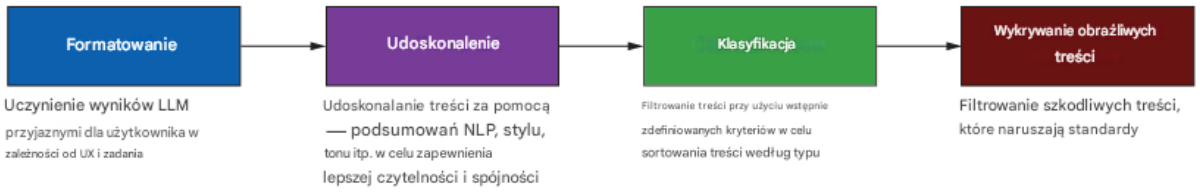
...

Obsługa modeli

Wiele nowoczesnych aplikacji AI jest hostowanych na platformach chmurowych ze względu na ich skalowalność i szeroki zakres oferowanych usług. Integracja z głównymi dostawcami chmury, takimi jak Microsoft Azure, Amazon Web Services czy Google Cloud Platform, umożliwia programistom korzystanie z bezpiecznej globalnej sieci centrów danych, usług zarządzanych ML oraz narzędzi do monitorowania i zarządzania aplikacjami. W związku z tym wiele przedsiębiorstw korzysta z jednego z modeli LLM hostowanych w chmurze, który jest udostępniany za pośrednictwem interfejsu API. Oznacza to, że dostawcy chmury zarządzający modelem służą do skalowania w górę lub w dół wnioskowania modelu. Jeśli niektóre modele są hostowane lokalnie, warstwa musi obsługiwać operacje modelu współpracujące z LLMOps. Architektura warstwy modelu powinna zapewniać strategiczne ramy do korzystania z wielu modeli LLM w celu stworzenia solidnej, wszechstronnej i skalowalnej aplikacji. Wymaga to starannego planowania wyboru modelu i zarządzania API, przy jednoczesnym zapewnieniu bezpieczeństwa i zgodności w zakresie przetwarzania danych. Architektura powinna być wystarczająco elastyczna, aby dostosowywać się do nowych modeli i interfejsów API w miarę ich pojawiania się.

Filtrowanie odpowiedzi

W większości przypadków aplikacja nie powinna udostępniać użytkownikowi końcowemu danych wygenerowanych z modelu; Powinien przejść przez etap przetwarzania, aby pomóc w zarządzaniu i filtrowaniu wszelkich wrażliwych szczegółów — w tym miejscu pomaga warstwa przetwarzania, a jej kluczowym zadaniem jest zarządzanie wynikami LLM. Warstwa filtrowania odpowiedzi ma za zadanie zapewnienie jakości i moderację treści, co jest kluczowe dla utrzymania zaufania do aplikacji GenAI. Obejmuje to użycie klasyfikatorów i narzędzi NLP do przesiewania wyników pod kątem dokładności, stronniczości i adekwatności. Jak widzieliśmy, wyniki LLM mogą się znacznie różnić, od prostego tekstu po złożone struktury danych. Zarządzanie tymi wynikami wymaga systematycznego podejścia, aby spełniały one standardy aplikacji i były prezentowane użytkownikowi w użytecznym formacie. Te kroki postprocessingu obejmują kilka obszarów, jak pokazano na rysunku 7.



Moderacja treści opiera się na praktykach RAI (Rail-Intelligent Intelligence), aby zminimalizować potencjalne ryzyko związane z generatywnymi modelami sztucznej inteligencji, takie jak treści stronnicze, obraźliwe lub wprowadzające w błąd, cyberzagrożenia, prywatność, kwestie prawne, wydajność i własność intelektualna. Musimy wdrożyć praktyki RAI, aby wykorzystać potencjał generatywnej sztucznej inteligencji. RAI jest niezbędne dla warstwy przetwarzania wyjściowego, aby sprostać ryzyku zarówno na poziomie aplikacji, jak i przedsiębiorstwa, takie jak wymogi regulacyjne i zgodności. Ponadto RAI może poprawić inne aspekty, takie jak prywatność, wyjaśnialność i uczciwość. Istnieje wiele narzędzi i frameworków, od których można zacząć. Na przykład InterpretML i Fairlearn firmy Microsoft to zestawy narzędzi open source, które pomagają programistom wyjaśniać i poprawiać uczciwość modeli uczenia maszynowego. AI Fairness 360 firmy IBM to kolejny zestaw narzędzi open source, który pomaga wykrywać i redukować stronniczość w modelach uczenia maszynowego. RAI omówimy bardziej szczegółowo w dalszej części książki. Dane wyjściowe i postprocessing mają kluczowe znaczenie dla zapewnienia użyteczności i bezpieczeństwa treści generowanych przez LLM. Architektura powinna zapewniać solidne ramy do udoskonalania i zarządzania danymi wyjściowymi, w tym formatowania, klasyfikacji treści, walidacji i buforowania. Zapewnienie jakości, zarówno zautomatyzowane, jak i sterowane przez użytkownika, musi być integralną częścią procesu, aby utrzymać wysokie standardy i doskonalić je z czasem. Ten rozdział pokazuje, jak GenAI można zintegrować z aplikacjami korporacyjnymi za pomocą nowego stosu aplikacji GenAI i powiązanej architektury aplikacji. Omówiliśmy również rolę Centrum Doskonałości w ułatwianiu tej integracji i rozwiązywaniu związanych z nią wyzwań technicznych, kulturowych i etycznych. Jednak zbudowanie rozwiązania AI to dopiero pierwszy krok; wdrożenie go w środowisku produkcyjnym i na dużą skalę wymaga innych umiejętności i narzędzi. W następnym rozdziale omówimy, co jest potrzebne do operacjonalizacji generatywnych rozwiązań AI i zapewnienia ich niezawodności, wydajności i bezpieczeństwa. Przyjrzymy się również najlepszym praktykom i ramom zarządzania cyklem życia AI oraz dostarczania wartości użytkownikom końcowym i interesariuszom.

Podsumowanie

* Copilot demonstruje, jak generatywna architektura AI może budować aplikacje i rozwiązania korporacyjne. Wykorzystuje inny stos aplikacji, który współpracuje z copilotami, aby stworzyć nowy stos architektury korporacyjnej. Ten stos jest przeznaczony dla aplikacji GenAI, które używają Copilot jako odpowiednika stosu LAMP.

* Stos aplikacji GenAI składa się z czterech warstw, które współpracują ze sobą, aby zapewnić funkcjonowanie stosu aplikacji — warstwy modelu, orkiestracji, uziemienia i filtrowania odpowiedzi.

* Warstwa orkiestracji jest jednym z kluczowych i fundamentalnych komponentów stosu GenAI. Obsługuje i organizuje różne procesy, usługi AI i platformy, aby zapewnić niezawodne i spójne działanie.

* Obszar frameworków orkiestracji jest nowy i ewoluuje, z wieloma zmianami i innowacjami. Niektóre z frameworków, które są obecnie szerzej stosowane, to SK, LangChain i LlamaIndex.

* Dzięki wykorzystaniu wtyczek, które potrafią obsłużyć zawiłości systemów źródłowych, ich protokołów i innych szczegółów, warstwa bazowa ułatwia integrację danych i wstępne przetwarzanie wdrożeń RAG w przedsiębiorstwach. Nadzoruje również osadzenia i powiązane bazy danych wektorowych.

* Warstwa modelu oferuje platformę do korzystania z wielu modeli z różnych źródeł — od modeli zarządzanych i precyzyjnie dostrojonych po modele BYOM (ang. Bring Your Own Model) dla

przedsiębiorstw. Dostęp do wszystkich tych modeli można uzyskać za pośrednictwem zaawansowanych interfejsów API, które gwarantują zgodność i bezpieczeństwo.

* Warstwa filtrowania odpowiedzi zapewnia jakość i moderuje treści, co jest niezbędne do budowania zaufania do aplikacji GenAI. Ponadto obejmuje ona wykorzystanie klasyfikatorów i narzędzi NLP do sprawdzania poprawności, rzetelności i adekwatności wyników.

* Centrum Doskonałości AI może pomóc przedsiębiorstwom w kompleksowej integracji programów LLM i GenAI z ich aplikacjami. Rozwiązując problemy techniczne, kulturowe i etyczne, przedsiębiorstwa mogą wykorzystać sztuczną inteligencję do zwiększenia innowacyjności i konkurencyjności, zapewniając sobie trwały sukces w świecie coraz bardziej opartym na sztucznej inteligencji.