Skalowanie: Najlepsze praktyki wdrażania produkcyjnego

Kiedy organizacje są gotowe przenieść swoje generatywne modele AI z etapu proof of concept (PoC) do rzeczywistego świata produkcji, rozpoczynają podróż, która wymaga starannego rozważenia kluczowych aspektów. Omówimy opcje wdrażania i skalowania, dzieląc się najlepszymi praktykami w zakresie zapewnienia operacyjności, niezawodności, wydajności i bezpieczeństwa generatywnych rozwiązań AI. Wdrażanie i skalowanie generatywnych modeli AI w środowisku produkcyjnym to złożone zadanie, które wymaga skrupulatnego rozważenia różnych czynników. Chociaż tworzenie PoC może być ekscytującym sposobem na przetestowanie wykonalności pomysłu, przeniesienie go do środowiska produkcyjnego wprowadza zupełnie nowy obszar zagadnień operacyjnych, technicznych i biznesowych. W tym rozdziale skupimy się na kluczowych aspektach, które deweloperzy muszą wziąć pod uwagę podczas wdrażania i skalowania generatywnych modeli AI w środowisku produkcyjnym. Omówimy kryteria operacyjne kluczowe dla monitorowania stanu systemów, opcje wdrożenia oraz najlepsze praktyki zapewniające niezawodność, wydajność i bezpieczeństwo. Zagłębimy się również w koncepcje operacji dużego modelu językowego (LLMOps) i operacji uczenia maszynowego (MLOps), które są niezbędne i ułatwiają zarządzanie cyklem życia generatywnych modeli AI w środowisku produkcyjnym. Ponadto rozdział podkreśli znaczenie zarządzania kosztami i budżetowania modeli wdrażanych w środowisku produkcyjnym oraz przedstawi kilka pouczających studiów przypadków udanego wdrożenia i skalowania generatywnych modeli AI w środowisku produkcyjnym. Pod koniec tego rozdziału przeżyjesz transformacyjną podróż, poznając kluczowe zagadnienia i najlepsze praktyki dotyczące wdrażania generatywnych modeli AI w środowisku produkcyjnym. Zanurzmy się w ten ekscytujący świat wiedzy, analizując niektóre z wyzwań, z jakimi boryka się większość przedsiębiorstw podczas wdrażania aplikacji GenAI w środowisku produkcyjnym.

Wyzwania związane z wdrożeniami produkcyjnymi

Generatywne aplikacje AI w korporacyjnym środowisku produkcyjnym napotykają specyficzne wyzwania, różniące się od tych w konwencjonalnym uczeniu maszynowym (ML). Jednak niektóre z wyzwań pozostają takie same. Na przykład, programiści muszą radzić sobie ze skomplikowaną relacją między wymaganiami dotyczącymi zasobów obliczeniowych, standardami jakości danych, celami wydajnościowymi, możliwością zmienności wyników oraz zmieniającą się sytuacją bezpieczeństwa wokół tych zaawansowanych modeli. Jednym z głównych wyzwań związanych z wdrażaniem generatywnych modeli AI jest ich złożoność. Modele te mogą być intensywne obliczeniowo i wymagać znacznych zasobów do szkolenia i wdrażania, nawet przy uwzględnieniu dzisiejszej infrastruktury i mocy obliczeniowych w chmurze. W związku z tym skalowanie modeli w celu obsługi dużej liczby żądań lub wdrażanie ich w środowiskach o ograniczonych zasobach może być trudne. Programiści muszą starannie rozważyć wymagania sprzętowe i programowe modeli, a także infrastrukturę niezbędną do ich obsługi, aby zapewnić ich efektywne wdrożenie i skalowanie. Kolejnym wyzwaniem związanym z wdrażaniem generatywnych modeli AI jest zapewnienie jakości i dostępności danych. Kluczowym aspektem jakości danych jest również znajomość źródła danych oraz tego, czy jest ono wiarygodne, czy autentyczne, co jest istotne. Modele te w dużym stopniu opierają się na jakości i dostępności danych, a wszelkie problemy z danymi mogą znacząco wpłynąć na wydajność i dokładność modeli. Programiści muszą wdrożyć solidne procesy walidacji i kontroli jakości danych oraz monitorować źródła danych i potoki wykorzystywane do trenowania i wdrażania modeli, aby zapewnić dokładność, trafność i aktualność danych. Można to osiągnąć, mierząc dokładność za pomocą predykcyjnych metryk wydajności, trafność poprzez oceny specyficzne dla danego zadania oraz aktualność poprzez śledzenie aktualności danych. Przedsiębiorstwa powinny wdrożyć solidne systemy monitorowania i dokumentować pochodzenie danych, aby utrzymać wysokie standardy integralności danych. Wydajność i dokładność modelu są również kluczowymi czynnikami przy wdrażaniu generatywnych

modeli AI. Deweloperzy muszą uważnie monitorować wydajność i dokładność modeli oraz wdrażać regularne procesy testowania i walidacji, aby zapewnić ich działanie zgodnie z oczekiwaniami. W idealnym świecie wymaga to dogłębnego zrozumienia algorytmów i architektur leżących u podstaw modeli oraz umiejętności diagnozowania i rozwiązywania wszelkich ewentualnych problemów. Jednak w praktyce większość przedsiębiorstw będzie dysponować wielofunkcyjnym zespołem programistów, analityków danych i ekspertów biznesowych, którzy wspólnie pomogą zrozumieć, ukierunkować i modelować architekturę oraz kwestie związane z wdrożeniem. Niezawodność i dostępność są również kluczowymi czynnikami przy wdrażaniu generatywnych modeli AI. Modele te muszą być niezawodne i dostępne, aby sprostać potrzebom firmy, co wymaga starannego rozważenia takich czynników, jak redundancja, przełączanie awaryjne i odzyskiwanie po awarii. Deweloperzy muszą wdrożyć solidne procesy monitorowania i konserwacji, aby zapewnić, że modele działają zgodnie z oczekiwaniami i być gotowi do szybkiego reagowania na wszelkie problemy. Oczywiście większość przedsiębiorstw polega na używanym przez siebie hiperskalerze, aby świadczyć znaczną część tej usługi. Niezawodność i dostępność tych usług są ściśle powiązane z dostawcami. W przypadku małych modeli językowych (SLM) i ich stosowania z dużymi modelami językowymi (LLM), kwestie niezawodności i skali są różne, zwłaszcza w przypadku wdrożeń brzegowych dla SLM. Bezpieczeństwo i zgodność z przepisami są również kluczowymi kwestiami. Modele te mogą przetwarzać wrażliwe dane, które muszą być chronione przed nieautoryzowanym dostępem, kradzieżą lub niewłaściwym wykorzystaniem. Przedsiębiorstwa muszą zapewnić zgodność modeli z odpowiednimi przepisami i standardami, takimi jak RODO, HIPAA lub PCI-DSS, oraz wdrożyć solidne mechanizmy kontroli bezpieczeństwa w celu ochrony danych i samych modeli. Firmy muszą najpierw poznać wymagania każdego z przepisów, aby zachować zgodność z przepisami o ochronie danych. Obejmuje to zarządzanie zgodami, zabezpieczanie poufnych informacji i reagowanie na naruszenia danych. Powinny śledzić i kontrolować dane osobowe wykorzystywane przez LLM, stosować silne środki bezpieczeństwa i uwzględniać prywatność w projekcie systemu od samego początku. Częste audyty zgodności, szkolenia pracowników i zarządzanie dostawcami są ważne dla utrzymania standardów. Dobry plan reagowania na incydenty w przypadku naruszeń danych i staranne prowadzenie dokumentacji pomogą w zachowaniu zgodności. Ponadto korzystanie z wbudowanych funkcji zgodności usług chmurowych może pomóc w spełnieniu tych wymagań. Przestrzegając standardów zgodności i podejmując te kroki, przedsiębiorstwa mogą wykorzystywać modele LLM do spełniania wymogów prawnych i regulacyjnych. Zarządzanie kosztami to kolejny ważny aspekt. Wdrożenie i utrzymanie modeli może być kosztowne, szczególnie jeśli chodzi o zasoby obliczeniowe, pamięci masowej i sieciowe. Deweloperzy muszą starannie zarządzać kosztami związanymi z wdrażaniem i skalowaniem modeli oraz być gotowi na kompromisy między kosztami a wydajnością w razie potrzeby. Integracja istniejących systemów i przepływów pracy ma również kluczowe znaczenie dla wdrażania generatywnych modeli AI. Modele te często wymagają integracji z istniejącymi systemami i przepływami pracy, co może być złożone i czasochłonne. Deweloperzy muszą zapewnić, że modele są kompatybilne z istniejącymi systemami i można je łatwo zintegrować z istniejącymi przepływami pracy. Muszą być również gotowi do ścisłej współpracy z innymi zespołami i interesariuszami, aby zapewnić płynne wdrożenie. Kolejnym ważnym czynnikiem jest udział człowieka w procesie. Modele te często wymagają interwencji lub nadzoru ze strony człowieka, szczególnie gdy są wykorzystywane do podejmowania kluczowych decyzji lub generowania treści wymagających weryfikacji przez człowieka. Deweloperzy muszą zadbać o to, aby modele były projektowane z uwzględnieniem czynnika ludzkiego i wdrażać solidne procesy zarządzania i monitorowania interwencji człowieka. Kwestie etyczne stanowią ostatni ważny czynnik przy wdrażaniu generatywnych modeli AI. Modele te mogą mieć istotne implikacje etyczne, szczególnie w odniesieniu do stronniczości, uczciwości i przejrzystości. Dlatego deweloperzy muszą zadbać o to, aby modele były projektowane i wdrażane etycznie i być przygotowani na rozwiązywanie problemów etycznych. Rozumiejąc te wyzwania i kwestie, deweloperzy mogą projektować i wdrażać generatywne modele AI, które są

skalowalne, niezawodne i bezpieczne oraz spełniają potrzeby firmy w środowisku produkcyjnym. Podczas wdrażania generatywnych modeli AI w środowisku produkcyjnym należy uwzględnić szereg wyzwań i kwestii, aby zapewnić pomyślną implementację. Poniższe kluczowe punkty podkreślają te kluczowe aspekty:

* Złożoność generatywnych modeli AI — wysokie wymagania obliczeniowe i znaczne zasoby są niezbędne do szkolenia i wdrożenia. Należy wziąć pod uwagę sprzęt, oprogramowanie i infrastrukturę, aby zapewnić efektywne skalowanie.

* Jakość i dostępność danych — są one niezbędne dla wydajności i dokładności modelu. Wdrożyć solidne procesy walidacji i kontroli jakości danych oraz monitorować źródła danych.

* Wydajność i dokładność modelu — wymagane są regularne testy i walidacja. Zespoły międzyfunkcyjne mogą pomóc w zrozumieniu i rozwiązywaniu problemów.

* Niezawodność i dostępność — wdrożyć redundancję, przełączanie awaryjne i odzyskiwanie po awarii. Stosować solidne procesy monitorowania i konserwacji. Niezawodność usług zależy od rozwiązań hiperskalerowych.

* Bezpieczeństwo i zgodność — chronić poufne dane przed nieautoryzowanym dostępem. Zapewnić zgodność z przepisami takimi jak RODO, HIPAA i PCI-DSS. Wdrożyć kontrole bezpieczeństwa i skutecznie zarządzać ochroną danych.

* Zarządzanie kosztami — obejmuje to staranne zarządzanie kosztami obliczeniowymi, pamięci masowej i sieci, równoważąc koszty i wydajność.

* Integracja z istniejącymi systemami — zapewnić kompatybilność i płynną integrację z obecnymi systemami i przepływami pracy. Współpracuj z innymi zespołami i interesariuszami.

* Kwestie związane z udziałem człowieka w procesie — Projektuj modele z nadzorem ludzkim w przypadku kluczowych decyzji. Wdrażaj procesy zarządzania interwencją człowieka.

* Kwestie etyczne — Rozwiązuj problemy związane z uprzedzeniami, uczciwością i przejrzystością. Zapewnij etyczne projektowanie i wdrażanie modeli.

Opcje wdrożenia

Wdrażanie generatywnych aplikacji AI wiąże się z kilkoma dostępnymi opcjami, a najlepszy wybór zależy od takich czynników, jak rozmiar i złożoność modelu, pożądana skalowalność i dostępność oraz dostępna infrastruktura i zasoby. Wdrożenie w chmurze oferuje korzyści, takie jak skalowalność, zróżnicowane opcje obliczeniowe i usługi zarządzane, ułatwiające wdrażanie. Należy jednak wziąć pod uwagę potencjalne koszty bieżące, uzależnienie od dostawcy oraz obawy dotyczące prywatności danych. Wdrożenie lokalne zapewnia większą kontrolę, optymalizację wydajności i bezpieczeństwo danych, ale wymaga znacznych początkowych inwestycji i wewnętrznej wiedzy specjalistycznej oraz może wiązać się z wolniejszym skalowaniem. Podejście hybrydowe łączy obie te zalety, umożliwiając przechowywanie wrażliwych danych lokalnie, jednocześnie wykorzystując skalowalność chmury i wprowadzając złożoność zarządzania. Niezależnie od wybranej ścieżki wdrożenia, kilka kluczowych technologii ułatwia ten proces. Konteneryzacja zapewnia spójne wykonywanie modelu w różnych środowiskach, a funkcje bezserwerowe idealnie nadają się do dynamicznych obciążeń. Bramy API zapewniają ustrukturyzowany dostęp dla innych aplikacji w celu wykorzystania modeli, a wyspecjalizowane platformy GenAI mogą usprawnić wdrażanie i zarządzanie modelami LLM. Wdrożenia w chmurze cieszą się popularnością ze względu na skalowalność i elastyczność, szczególnie w przypadku dostawców takich jak Microsoft Azure, Amazon Web Services (AWS) i Google Cloud

Platform (GCP). W zależności od potrzeb, programiści mogą wybierać między maszynami wirtualnymi, kontenerami lub funkcjami bezserwerowymi. Kluczowe jest jednak dokładne oszacowanie wymaganej infrastruktury i zasobów, w tym procesorów graficznych (GPU), pamięci, pamięci masowej i przepustowości sieci. Wdrożenie strategii równoważenia obciążenia i redundancji zapewnia skalowalność i dostępność, a solidne monitorowanie i automatyczne testowanie są niezbędne do utrzymania wydajności i sprawności systemu. Starannie uwzględniając te czynniki, programiści mogą zapewnić niezawodne, skalowalne i ekonomiczne wdrażanie generatywnych aplikacji AI, niezależnie od wybranego środowiska.

Zarządzane programy LLM za pośrednictwem API

Oprócz omówionych wcześniej opcji wdrażania, należy zauważyć, że niektóre programy LLM są dostępne wyłącznie za pośrednictwem zarządzanego interfejsu API hostowanego online. Często ma to miejsce w przypadku najnowocześniejszych modeli opracowywanych przez organizacje badawcze zajmujące się AI lub duże firmy technologiczne. Jak wiemy, modele GenAI wymagają znacznych zasobów obliczeniowych, co utrudnia ich uruchomienie lokalnie lub w sposób hybrydowy. Tabela przedstawia niektóre zalety zarządzanych modeli LLM.

Zalety: Opis

Łatwość użycia: Zarządzane programy LLM za pośrednictwem API są zazwyczaj łatwe w użyciu. Programiści mogą wysyłać żądania do API i otrzymywać odpowiedzi, nie martwiąc się o infrastrukturę bazową ani złożoność modelu.

Ciągłe aktualizacje: Dostawcy zarządzanych programów LLM często stale aktualizują i ulepszają swoje modele. API pozwala korzystać z tych ulepszeń bez konieczności ręcznej aktualizacji modeli.

Skalowalność: Zarządzane modele LLM za pośrednictwem API mogą obsługiwać dużą liczbę żądań i skalować się automatycznie w zależności od zapotrzebowania, podobnie jak inne usługi chmurowe.

Złożoność modelu: Modele LLM to niezwykle złożone modele uczenia maszynowego, które mogą stanowić wyzwanie dla przedsiębiorstw, szczególnie tych bez dużego doświadczenia w dziedzinie sztucznej inteligencji i uczenia maszynowego. Usługi zarządzane przenoszą tę złożoność na dostawcę, udostępniając wnioskowanie za pośrednictwem API.

Istnieją również pewne ograniczenia i wyzwania, które należy wziąć pod uwagę podczas korzystania z zarządzanych LLM za pośrednictwem interfejsu API, jak przedstawiono w tabeli 2.

Zagadnienia: Opis

Koszt: Koszt korzystania z zarządzanego LLM za pośrednictwem API może się znacznie różnić w zależności od sposobu użytkowania. Chociaż niektórzy dostawcy oferują bezpłatne poziomy, bardziej rozbudowane użytkowanie może wiązać się ze znacznymi kosztami.

Zależność: Korzystając z zarządzanego LLM za pośrednictwem API, jesteś zależny od dostawcy w zakresie modelu i infrastruktury. Jeśli dostawca doświadczy awarii lub zaprzestanie świadczenia usługi, może to wpłynąć na Twoją aplikację.

Prywatność danych: Dane są przesyłane na serwery dostawcy w celu przetworzenia za pomocą zarządzanego LLM za pośrednictwem API, co może budzić obawy dotyczące prywatności, zwłaszcza w odniesieniu do danych wrażliwych.

Ograniczone możliwości personalizacji: Chociaż zarządzane LLM za pośrednictwem API są łatwe w użyciu, zazwyczaj oferują ograniczone możliwości personalizacji. Jesteś ograniczony do możliwości i konfiguracji udostępnianych przez API i nie możesz modyfikować bazowego modelu.

Podsumowując, zarządzane modele LLM za pośrednictwem API oferują szereg korzyści, ale wiążą się z nimi również pewne kwestie. To, czy są one odpowiednią opcją dla Twojej aplikacji GenAI, zależy od Twoich potrzeb i ograniczeń. Jeśli potrzebujesz wysokiego poziomu personalizacji, masz surowe wymagania dotyczące prywatności danych lub musisz uruchomić model offline, wdrożenie lokalne lub hybrydowe może być bardziej odpowiednie. Zarządzany model LLM za pośrednictwem API może być jednak dobrym wyborem, jeśli cenisz sobie łatwość obsługi, ciągłe aktualizacje i automatyczne skalowanie.

Najlepsze praktyki wdrażania produkcyjnego

Aby korzystać z aplikacji GenAI, wymagane jest kompleksowe podejście, obejmujące staranne planowanie i realizację w celu zapewnienia skalowalności, niezawodności i bezpieczeństwa. Korzystając z modeli LLM w swojej aplikacji, musisz wziąć pod uwagę takie aspekty, jak LLMOps, obserwowalność i narzędzia, aby skutecznie zarządzać cyklem życia aplikacji. Ponadto musisz wziąć pod uwagę inne aspekty, takie jak obsługa i zarządzanie modelami, kwestie niezawodności i wydajności oraz kwestie bezpieczeństwa i zgodności. Obszary te są ważne, ponieważ pozwalają mieć pewność, że aplikacja działa tak, jak powinna, i spełnia wysokie standardy niezawodności, bezpieczeństwa i zgodności.

Podsumowując, zarządzane modele LLM za pośrednictwem API oferują szereg korzyści, ale wiążą się z nimi również pewne kwestie. To, czy są one odpowiednią opcją dla Twojej aplikacji GenAI, zależy od Twoich potrzeb i ograniczeń. Jeśli potrzebujesz wysokiego poziomu personalizacji, masz surowe wymagania dotyczące prywatności danych lub musisz uruchomić model offline, wdrożenie lokalne lub hybrydowe może być bardziej odpowiednie. Zarządzany model LLM za pośrednictwem API może być jednak dobrym wyborem, jeśli cenisz sobie łatwość obsługi, ciągłe aktualizacje i automatyczne skalowanie.

Metryki wnioskowania LLM

Jedna z najważniejszych metryk z perspektywy wdrożenia produkcyjnego jest związana z wnioskowaniem LLM. Jest to główny obszar, nad którym wszyscy pracujemy i z którym mamy do czynienia podczas tworzenia aplikacji GenAI. Jak widzieliśmy, LLM generują tekst w dwóch krokach: monicie, gdzie tokeny wejściowe są przetwarzane jednocześnie, oraz dekodowaniu, gdzie tekst jest tworzony sekwencyjnie, jeden token na raz. Każdy utworzony token jest dodawany do danych wejściowych i ponownie wykorzystywany przez model do utworzenia kolejnego tokena. Generowanie kończy się, gdy LLM wygeneruje specjalny token stopu lub gdy spełniony zostanie warunek zdefiniowany przez użytkownika (np. zostanie wygenerowana maksymalna liczba tokenów). Zrozumienie i zarządzanie kluczowymi metrykami operacyjnymi związanymi z wnioskowaniem LLM staje się kluczowe. Wiele z tych metryk jest nowych i wciąż zbyt wczesnych, aby większość użytkowników mogła się z nimi zaznajomić, ale następujące cztery metryki są szczególnie ważne: czas do pierwszego tokena, czas na token wyjściowy, opóźnienie i przepustowość. Tabela 3 przedstawia definicję i znaczenie tych kryteriów operacyjnych. W dalszej części rozdziału dowiesz się, jak mierzyć te parametry w naszym wdrożeniu LLM.

Metryka: Definicja

Czas do pierwszego tokena (TTFT): Mierzy czas potrzebny modelowi na wygenerowanie pierwszego tokena po zapytaniu użytkownika. Niższy TTFT oznacza bardziej responsywne środowisko użytkownika. Na TTFT wpływa czas potrzebny na przetworzenie monitu i wygenerowanie pierwszego tokena wyjściowego.

Czas na token wyjściowy (TPOT): Oblicza czas potrzebny modelowi na wygenerowanie jednego tokena dla określonego zapytania. Niższy TPOT oznacza szybsze generowanie tekstu. Rozmiar modelu, konfiguracja sprzętowa i algorytm dekodowania wpływają na TPOT.

Opóźnienie: Ta metryka mierzy czas potrzebny na przesłanie danych z punktu początkowego do miejsca docelowego. W przypadku modeli LLM jest to czas potrzebny modelowi na wygenerowanie odpowiedzi dla użytkownika. Model i wygenerowane tokeny wpływają na opóźnienie modeli LLM. Zazwyczaj większość czasu poświęcana jest na generowanie kompletnych tokenów, które są generowane pojedynczo. Im dłuższe generowanie, tym większe opóźnienie.

Przepustowość: Mierzy ilość danych, które można przesłać w jednostce czasu. W tym przypadku liczba tokenów wyjściowych na sekundę w jednostce wdrożeniowej może zostać obsłużona we wszystkich żądaniach.

Żądanie na sekundę (RPS): RPS mierzy przepustowość LLM w środowisku produkcyjnym i wskazuje liczbę żądań, które LLM może obsłużyć w ciągu sekundy. Ta metryka ma kluczowe znaczenie dla zrozumienia skalowalności i wydajności LLM wdrożonych w rzeczywistych aplikacjach.

UWAGA: RPS i przepustowość są często używane zamiennie w kontekście metryk wydajności, ale mogą się one różnić niuansami. Zasadniczo, RPS odnosi się do obciążenia przychodzącego, natomiast przepustowość odnosi się do danych wyjściowych serwera lub do jego skutecznego obsłużenia. Wysoka przepustowość przy wysokim RPS wskazuje na dobrze działający serwer, podczas gdy niska przepustowość przy wysokim RPS może sugerować, że serwer ma problemy z nadążaniem za zapotrzebowaniem.

Opóźnienie

Opóźnienie jest powszechnie używanym wskaźnikiem przez niemal wszystkich, ale jest niejasne i wymaga ponownego przeanalizowania w kontekście sztucznej inteligencji generatywnej. Powszechna definicja opóźnienia nie jest adekwatna, ponieważ interfejsy API zwracały tylko jeden wynik zamiast wielu odpowiedzi strumieniowych. Ponieważ generowanie danych wyjściowych w dużym stopniu zależy od danych wejściowych, GenAI musi brać pod uwagę różne punkty opóźnienia. Na przykład, jedno opóźnienie to opóźnienie pierwszego tokena; drugie to pełne opóźnienie od początku do końca po zakończeniu generowania.

Nie możemy polegać wyłącznie na drugim opóźnieniu od początku do końca, ponieważ wiemy, że rozmiar monitu i liczba tokenów wyjściowych są kluczowymi czynnikami wpływającymi. Generowanie zmienia się w zależności od zapytania (tj. monitu) — nie jest to użyteczna metryka, chyba że porównamy podobne tokeny. Na przykład, poniższe dwa wymagają różnych ilości obliczeń i czasu, nawet gdy tokeny wejściowe są mniej więcej takie same:

* Przykład 1 — Wygeneruj trzywersowy wiersz o tym, dlaczego psy są niesamowite.

* Przykład 2 — Wygeneruj trzystronicowy wiersz o tym, dlaczego psy są niesamowite.

Pierwszy przykład ma 11 tokenów, a drugi 10 tokenów przy użyciu tokenizatora cl100kbase (używanego przez nowsze modele GPT). Wygenerowane tokeny są jednak bardzo różne. Ponadto, jak opisano wcześniej, token czasu na wyjście (TPOT) nie uwzględnia monitu wejściowego. Monit

wejściowy jest również duży dla wielu zadań, takich jak podsumowanie, ponieważ generowanie rozszerzone o pobieranie (RAG) jest używane do informacji w kontekście. Zatem użycie TPOT jako sposobu pomiaru opóźnienia nie jest precyzyjne. Rozmiar modelu również wpływa na wykorzystanie zasobów; mniejszy model jest zazwyczaj bardziej wydajny i zużywa mniej zasobów, podczas gdy większy model jest bardziej wydajny i wydajny, ale zajmuje znacznie więcej czasu. Użyjmy przykładu, aby pokazać, jak to zmierzyć. Poniższy listing przedstawia prostą metodę pomiaru opóźnienia interfejsu API czatu Azure OpenAI. W przeciwieństwie do poprzednich przykładów, wykorzystujących zestawy narzędzi programistycznych (SDK), ten korzysta z interfejsów API REST, a zatem musimy skonstruować ładunek i wywołać metody POST. Wybieramy liczbę żądań do symulacji i tworzymy funkcję główną (main), która wykorzystuje ThreadPoolExecutor do jednoczesnego wysyłania kilku żądań API. Przekazuje ona funkcję call_api_and_measure_latency() do executora dla każdego symulowanego żądania, zbiera opóźnienia wszystkich żądań, oblicza średnie opóźnienie i wyświetla je.

Listing1 : Measuring latency

import os

...

```
AZURE_ENDPOINT = os.getenv("AOAI_ENDPOINT")
AOAI_API_KEY = os.getenv("AOAI_KEY")
MODEL = "gpt35"
API_VERSION = "2024-02-15-preview"

headers = {
    "api-key": AOAI_API_KEY,
    "Content-Type": "application/json"
}

def get_payload():
    return {
        "model": MODEL,
        "max_tokens": 50,
        "messages": [{"role": "system", "content":
                      "You are a helpful assistant."},
                     {"role": "user", "content": "Hello, world!"}],
        "temperature": 0.95,
        "stream": True
    }

def call_api_and_measure_latency():
    payload = get_payload()
    start_time = time.time()
    response = requests.post(AZURE_ENDPOINT,
                             headers=headers,
                             json=payload, timeout=20)
    latency = time.time() - start_time
    return latency, response.status_code

num_requests = 50

def main():
    with ThreadPoolExecutor(max_workers=20) as executor:
        futures = [executor.submit(call_api_and_measure_latency)
                   for _ in range(num_requests)]
        latencies = []
        for future in tqdm(as_completed(futures), total=num_requests):
            latency, status_code = future.result()
            print(f"Latency: {latency}s, Status Code: {status_code}")
            latencies.append(latency)

    average_latency = sum(latencies) / len(latencies)
    print(f"Average Latency: {average_latency}s")

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

Setting Azure OpenAI Chat API endpoint and API key

Defines the payload, including the model details to use

We stream the response so we can start getting the response faster.

Function to call the Azure OpenAI Chat API and measure latency

Start time used to calculate latency

End time used to calculate latency

Number of requests to simulate

Simulates concurrent API calls

Calculates and print latency metrics

Rysunek przedstawia przykład danych wyjściowych wykonanych z 50 iteracjami i średnim opóźnieniem 11,35 sekundy w instancji pay-as-you-go (PAYGO).

```
 64%|                                                                    | 32/50 [00:24<00:11,  1.55it/s]L
atency: 13.265730857849121s, Status Code: 200
 66%|                                                                    | 33/50 [00:25<00:10,  1.70it/s]L
atency: 13.375885963439941s, Status Code: 200
Latency: 13.342093706130981s, Status Code: 200
Latency: 13.474364042282104s, Status Code: 200
 72%|                                                                    | 36/50 [00:25<00:04,  2.89it/s]L
atency: 13.714608430862427s, Status Code: 200
Latency: 13.73707389831543s, Status Code: 200
Latency: 13.400543689727783s, Status Code: 200
Latency: 13.547471284866333s, Status Code: 200
Latency: 8.889911890029907s, Status Code: 200
 82%|                                                                    | 41/50 [00:28<00:03,  2.27it/s]L
atency: 8.495208740234375s, Status Code: 200
 84%|                                                                    | 42/50 [00:28<00:03,  2.15it/s]L
atency: 8.236473083496094s, Status Code: 200
 86%|                                                                    | 43/50 [00:29<00:03,  2.10it/s]L
atency: 8.861016273498535s, Status Code: 200
Latency: 8.165524005889893s, Status Code: 200
Latency: 8.456485033035278s, Status Code: 200
Latency: 8.338528156280518s, Status Code: 200
 94%|                                                                    | 47/50 [00:29<00:00,  3.80it/s]L
atency: 8.514750242233276s, Status Code: 200
Latency: 8.58434534072876s, Status Code: 200
 98%|                                                                    | 49/50 [00:29<00:00,  4.52it/s]L
atency: 8.795718669891357s, Status Code: 200
100%|                                                                    | 50/50 [00:29<00:00,  1.68it/s]
Average Latency: 11.354879903793336s
```

Jest to połączenie w obie strony od klienta do usługi, a nie opóźnienie samej usługi. Nie jest to wynik optymalny i w przypadku większości obciążeń produkcyjnych musimy zwrócić uwagę na zarezerwowaną pojemność, którą omówimy w następnej sekcji. Jak pokazano na rysunku2, w tym przykładzie możemy użyć gotowych funkcji platformy Azure do uzyskania metryk usługi, takich jak opóźnienie.

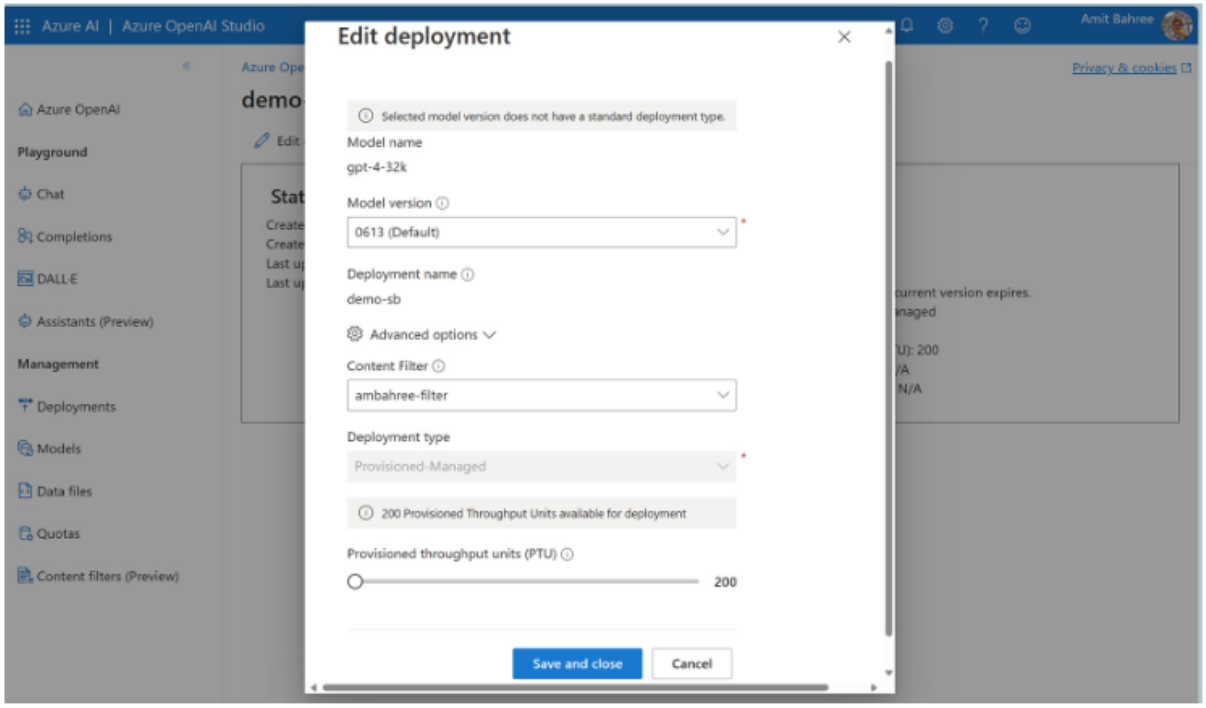
Używając domyślnych opcji metryk, widzimy średnie opóźnienie dla tej instancji PAYGO wynoszące 95,37 milisekundy.

UWAGA: Kod, który widzieliśmy wcześniej, to podstawowy przykład pokazujący, jak mierzyć opóźnienie i widok z perspektywy produkcyjnej; nie jest to dobra implementacja do testowania opóźnień w testach obciążeniowych, zwłaszcza jeśli nie korzysta się z PAYGO. Lepszym rozwiązaniem jest użycie skryptu z narzędziami OSS, takimi jak Apache JMeter.

Skalowalność

Jedną z głównych opcji skalowania, które przedsiębiorstwo powinno rozważyć podczas wdrażania aplikacji produkcyjnej korzystającej z platformy LLM, takiej jak Azure OpenAI, są aprowizowane jednostki przepływności (PTU). PTU dla Azure OpenAI to jednostki mocy obliczeniowej modelu, które

można zarezerwować i wdrożyć do przetwarzania monitów i generowania uzupełnień. Ucieleśniają one znormalizowany sposób reprezentowania przepustowości wdrożenia, przy czym każda para model-wersja wymaga różnych wartości dla wdrożenia i przepustowości przypadającej na PTU. Przepustowość przypadająca na PTU może się różnić w zależności od typu i wersji modelu, a znajomość tej wartości jest ważna dla prawidłowego skalowania aplikacji. Jednostka PTU jest zasadniczo tym samym, co zarezerwowana instancja dostępna w innych usługach platformy Azure, ale jest to tylko funkcja usługi OpenAI platformy Azure. Gdy aplikacja wymaga skalowania i korzysta z wielu usług AI, zarezerwowana pojemność instancji musi być uwzględniona we wszystkich tych usługach, ponieważ nie istnieje uniwersalna usługa rezerwująca pojemność dla konkretnej aplikacji lub subskrypcji. Aby wdrożyć model w usłudze Azure OpenAI przy użyciu jednostek PTU, należy wybrać typ wdrożenia „zarządzane aprowizowane" i wskazać liczbę jednostek PTU wymaganych dla obciążenia, jak pokazano na rysunku 3.



Musimy również obliczyć rozmiar naszych konkretnych kształtów obciążenia, co można zrobić za pomocą kalkulatora pojemności platformy Azure OpenAI. To obliczenie pomaga określić odpowiednią liczbę jednostek PTU dla danego wdrożenia. Oprócz jednostek PTU, przedsiębiorstwa mogą korzystać z modelu PAYGO, który wykorzystuje tokeny na minutę (TPM) zużywane na żądanie. Model ten można połączyć z jednostkami PTU w celu optymalizacji wykorzystania i kosztów. Ponadto, API Management (APIM) można wykorzystać z Azure OpenAI do zarządzania i wdrażania zasad kolejkowania, ograniczania przepustowości, obsługi błędów i limitów wykorzystania. Przeprowadzając te same testy opóźnień, co dla PAYGO na instancji PTU, z niewielkimi modyfikacjami, uzyskaliśmy następujące wyniki dla obu, przy użyciu GPT-4 i tej samej wersji modelu. Losowo wybieramy monit z listy do wywołania i powtarzamy 100 iteracji w każdym przypadku. Średnie opóźnienie między końcami na jednostkach PTU wynoszące 2,9 sekundy jest całkiem przyzwoite w porównaniu z 6,3 sekundy w PAYGO, co jest wartością niezłą, ale niezbyt imponującą:

Starting PTU test...

Median Latency: 1.582270622253418s

Average Latency: 2.947581880092621s

Min Latency: 0.7084167003631592s

Max Latency: 11.790298700332642s

Starting PAYGO test...

Median Latency: 2.391003727912903s

Average Latency: 6.372000885009766s

Min Latency: 0.4583735466003418s

Max Latency: 89.96037220954895s

Kod na listingu 2 pokazuje różnicę. Ta funkcja iteruje po dwóch klientach OpenAI i odpowiadających im modelach. Dla każdej pary klient-model tworzony jest obiekt ThreadPoolExecutor z 20 procesami roboczymi, a następnie przesyłane są zadania. Każde zadanie to wywołanie funkcji call_completion_api() (wtyczki dla interfejsu API uzupełniania Azure OpenAI) z losowo wybranymi danymi wejściowymi z danych testowych. Funkcja zbiera opóźnienia wszystkich zadań, oblicza medianę, średnią, minimalne i maksymalne opóźnienie oraz wyświetla te metryki.

Listing 2: Pomiar opóźnienia między PAYGO a PTU

```
test_inputs = ["Hello", "How are you?",
➥"What's the capital of Hawaii?", "Tell me a dad joke",
➥"Tell me a story", "What's your favorite movie?",
➥"What's the meaning of life?", "What's the capital of India?",
➥"What's the square root of 1976?", "What's the largest mammal?",
➥"Write a story about a Panda F1 driver in less
➥than {MAX_TOKENS} words"]
def main():
for client, model, test_name in [(ptu_client,
➥PTU_MODEL, "PTU"), (paygo_client, PAYGO_MODEL, "PAYGO")]:
print(f"Starting {test_name} test...")
with ThreadPoolExecutor(max_workers=20) as executor:
latencies = []
futures = [executor.submit(call_completion_api,
➥client, model, input) for input in
➥random.choices(test_inputs, k=NUM_INTERATION)]
for future in tqdm(as_completed(futures),
```

```
➡total=NUM_INTERATION):
latency, token_count = future.result()
if latency is not None and token_count is not None:
logging.info(f"Latency: {latency}s,
➡Token Count: {token_count}")
latencies.append(latency)
# Calculate and print metrics
average_latency = sum(latencies) / len(latencies)
➡if latencies else None
min_latency = min(latencies) if latencies else None
max_latency = max(latencies) if latencies else None
median_latency = statistics.median(latencies)
➡if latencies else None
print(f"Median Latency: {median_latency}s")
print(f"Average Latency: {average_latency}s")
print(f"Min Latency: {min_latency}s")
print(f"Max Latency: {max_latency}s")
```

PAYGO

Model PAYGO z TPM to elastyczna metoda płatności, która pozwala płacić tylko za wykorzystane zasoby. Metoda ta jest szczególnie przydatna w przypadku aplikacji o zmiennych wzorcach wykorzystania i niewymagających stałej mocy obliczeniowej. Jest to standard dla większości klientów i aplikacji u większości dostawców. TPM to miara mocy obliczeniowej modelu. Gdy wysyłasz żądanie do modelu, używa on określonej liczby tokenów w oparciu o monit oraz złożoność i długość odpowiedzi. Opłaty naliczane są za każdy zużyty token, więc wraz ze wzrostem wykorzystania płacisz więcej, a wraz ze spadkiem – mniej. Większość chmurowych systemów LLM posiada funkcję zarządzania limitami, która pozwala przypisywać limity wydajności do wdrożeń do limitu globalnego. Podobnie, wdrożenie i limity wydajności są powiązane z wdrożeniem modelu. Możemy również przypisać konkretny moduł TPM do konkretnego wdrożenia; w takim przypadku dostępny limit dla tego modelu zostanie zmniejszony o tę wartość. Model PAYGO jest korzystny dla skalowania, ponieważ umożliwia globalną dystrybucję modułu TPM w ramach subskrypcji i regionu, zapewniając elastyczność w zarządzaniu alokacją limitów przepustowości pomiędzy wdrożeniami w ramach subskrypcji. Model ten idealnie sprawdza się w aplikacjach, w których szczytowe okresy intensywnego wykorzystania danych następują po okresach niskiego lub zerowego wykorzystania, ponieważ gwarantuje, że płacisz tylko za to, z czego korzystasz.

Limity i limity przepustowości

Limity i limity przepustowości to dwa mechanizmy wykorzystywane w usługach chmurowych do zarządzania i kontrolowania wykorzystania zasobów. Limity to całkowita ilość zasobów, jaką użytkownik lub usługa może wykorzystać w określonym czasie, takim jak dzień lub miesiąc. Działają one jako limit, zapobiegając nadmiernemu zużyciu zasobów i zapewniając sprawiedliwy podział między użytkowników.

PAYGO

Model PAYGO z TPM to elastyczna metoda płatności, która pozwala płacić tylko za wykorzystane zasoby. Metoda ta jest szczególnie przydatna w przypadku aplikacji o zmiennych wzorcach wykorzystania i niewymagających stałej mocy obliczeniowej. Jest to standard dla większości klientów i aplikacji u większości dostawców. TPM to miara mocy obliczeniowej modelu. Gdy wysyłasz żądanie do modelu, używa on określonej liczby tokenów w oparciu o monit oraz złożoność i długość odpowiedzi. Opłaty naliczane są za każdy zużyty token, więc wraz ze wzrostem wykorzystania płacisz więcej, a wraz ze spadkiem – mniej. Większość chmurowych systemów LLM posiada funkcję zarządzania limitami, która pozwala przypisywać limity wydajności do wdrożeń do limitu globalnego. Podobnie, wdrożenie i limity wydajności są powiązane z wdrożeniem modelu. Możemy również przypisać konkretny moduł TPM do konkretnego wdrożenia; w takim przypadku dostępny limit dla tego modelu zostanie zmniejszony o tę wartość. Model PAYGO jest korzystny dla skalowania, ponieważ umożliwia globalną dystrybucję modułu TPM w ramach subskrypcji i regionu, zapewniając elastyczność w zarządzaniu alokacją limitów przepustowości pomiędzy wdrożeniami w ramach subskrypcji. Model ten idealnie sprawdza się w aplikacjach, w których występują okresy szczytowego obciążenia, po których następują okresy niskiego lub zerowego obciążenia, ponieważ gwarantuje, że płacisz tylko za to, z czego korzystasz.

Limity i limity przepustowości

Limity i limity przepustowości to dwa mechanizmy wykorzystywane w usługach chmurowych do zarządzania i kontrolowania wykorzystania zasobów. Limity to całkowita ilość zasobów, jaką użytkownik lub usługa może wykorzystać w określonym czasie, takim jak dzień lub miesiąc. Działają one jako limit wykorzystania, zapobiegając nadmiernemu zużyciu zasobów i zapewniając sprawiedliwy podział między użytkowników. Natomiast limity przepustowości kontrolują częstotliwość żądań do usługi. Zazwyczaj definiuje się je jako liczbę żądań, które można wysłać na sekundę lub minutę. Ograniczając częstotliwość, z jaką użytkownicy mogą wysyłać żądania, limity przepustowości pomagają zarządzać obciążeniem i zapobiegać przeciążaniu systemów. W istocie limity odnoszą się do ilości zasobów, z których można korzystać, podczas gdy limity szybkości odnoszą się do częstotliwości dostępu do tych zasobów. Zrozumienie obu tych kwestii jest kluczowe dla efektywnego zarządzania interfejsami API i unikania przerw w świadczeniu usług w przedsiębiorstwach. Przestrzegając limitów szybkości, przedsiębiorstwa mogą zapewnić, że ich aplikacje nie będą wysyłać większej liczby żądań, niż usługa może obsłużyć w danym momencie, co pomaga utrzymać wydajność i stabilność. Jednocześnie, utrzymując się w ramach limitów, mogą kontrolować koszty i zapobiegać nieoczekiwanym przekroczeniom limitów. Limity dla usługi OpenAI, a w szczególności dla Azure OpenAI, są definiowane jako limity zasobów lub mocy obliczeniowej, jaką może wykorzystać użytkownik lub organizacja. Kwoty te są zazwyczaj mierzone w TPM i przypisywane dla każdego regionu i modelu. Limity zapewniają, że usługa może utrzymać spójną i przewidywalną wydajność dla wszystkich użytkowników. Przedsiębiorstwa powinny traktować te limity jako sposób na efektywne zarządzanie wykorzystaniem i kosztami. Muszą monitorować zużycie, aby uniknąć przekroczenia tych limitów, co mogłoby prowadzić do dodatkowych opłat lub przerw w świadczeniu usług. Ważne jest również, aby przedsiębiorstwa rozumiały limity szybkości związane z wdrożeniami i odpowiednio planowały. Na przykład, jeśli przedsiębiorstwo ma limit 240 000 modułów TPM dla określonego

modelu w regionie, może utworzyć jedno wdrożenie z 240 tys. modułów TPM, dwa po 120 tys. modułów TPM każde lub wiele wdrożeń, których łączna liczba modułów TPM w danym regionie będzie mniejsza niż 240 tys. modułów. Na przykład, rysunek 4 przedstawia ustawienie limitu dla określonego punktu końcowego usługi Azure OpenAI i różnych wdrożonych modeli.

OpenAI ma swój system limitów, ale są one inaczej ustrukturyzowane. Limity OpenAI są zazwyczaj powiązane z limitami użytkowania, które są ustalane na podstawie danych rozliczeniowych podanych przez użytkownika. Po wprowadzeniu danych rozliczeniowych użytkownicy mają zatwierdzony limit użytkowania w wysokości określonej kwoty miesięcznie (domyślnie 100 USD), który może automatycznie wzrastać wraz ze wzrostem wykorzystania platformy. Użytkownicy przechodzą z jednego poziomu użytkowania do drugiego, jak pokazano na rysunku 5. Użytkownicy mogą sprawdzić swój aktualny limit użytkowania w ustawieniach konta na stronie limitów.

TIER	QUALIFICATION	USAGE LIMITS
Free	User must be in an allowed geography	$100 / month
Tier 1	$5 paid	$100 / month
Tier 2	$50 paid and 7+ days since first successful payment	$500 / month
Tier 3	$100 paid and 7+ days since first successful payment	$1,000 / month
Tier 4	$250 paid and 14+ days since first successful payment	$5,000 / month
Tier 5	$1,000 paid and 30+ days since first successful payment	$10,000 / month

Select a tier below to view a high-level summary of rate limits per model.

Free Tier 1 Tier 2 Tier 3 Tier 4 Tier 5

Limity te mają na celu ułatwienie zarządzania kosztami i ich przewidywania oraz zapobieganie nadmiernemu wykorzystaniu zasobów. Przedsiębiorstwa powinny uważnie monitorować swoje wykorzystanie, aby upewnić się, że nie przekracza ono tych limitów i zrozumieć, jak limity te mogą się skalować wraz ze wzrostem wykorzystania.

Zarządzanie limitami

Efektywne zarządzanie limitami ma kluczowe znaczenie dla utrzymania spójnej i przewidywalnej wydajności aplikacji. Oto kilka najlepszych praktyk, które warto rozważyć:

* Zrozum swoje limity. Zapoznaj się z domyślnymi limitami i limitami obowiązującymi dla poszczególnych modeli, ponieważ każdy model i region może mieć inne domyślne limity limitów.

* Monitoruj swoje wykorzystanie. Wdróż strategie monitorowania, aby śledzić wykorzystanie w odniesieniu do przypisanych limitów. Pomoże to uniknąć nieoczekiwanego ograniczania przepustowości i zapewnić klientom dobre doświadczenia.

* Wdróż logikę ponawiania prób. W swojej aplikacji uwzględnij logikę ponawiania prób, aby obsługiwać błędy związane z limitami przepustowości. Pozwoli to aplikacji czekać i ponawiać żądanie po krótkiej przerwie, zamiast od razu kończyć je niepowodzeniem. Prostym sposobem na to jest skorzystanie z biblioteki Tenacity (biblioteki OSS):

```
from tenacity import (

retry,

stop_after_attempt

wait_random_exponential,

@retry(wait=wait_random_exponential(min=1, max=60),

stop=stop_after_attempt(6))

def completion_backoff(conversation):

response = client.chat.completions.create(

model=MODEL,

messages=conversation,

temperature=TEMPERATURE,

max_tokens=MAX_TOKENS,

)

return response
```

* Unikaj gwałtownych zmian obciążenia. Stopniowo zwiększaj obciążenie, aby zapobiec nagłym skokom, które mogą prowadzić do dławienia przepustowości. Przetestuj różne schematy wzrostu obciążenia, aby znaleźć najefektywniejsze podejście dla swojej aplikacji. Należy pamiętać, że dławienie przepustowości celowo spowalnia lub ogranicza liczbę żądań, które aplikacja lub usługa może obsłużyć w określonym czasie. Serwer lub dostawca usług zazwyczaj wymusza to, aby zapobiec przeciążeniu systemu, zapewnić sprawiedliwe wykorzystanie i utrzymać jakość usług. Jak wiemy, dławienie przepustowości jest powszechną praktyką w zarządzaniu API i usługach chmurowych, służącą do efektywnego zarządzania zasobami i ochrony systemu przed potencjalnymi nadużyciami lub atakami typu „odmowa usługi” (DoS). Służy ono również do zapobiegania zużywaniu przez pojedynczego użytkownika lub usługę wszystkich dostępnych zasobów i wpływaniu na wydajność innych użytkowników lub usług.

* Zarządzaj alokacją modułu TPM. Użyj funkcji zarządzania limitami, aby zwiększyć limit TPM we wdrożeniach o dużym natężeniu ruchu i zmniejszyć limit TPM we wdrożeniach o ograniczonych potrzebach. Pomaga to zrównoważyć obciążenie i zoptymalizować wykorzystanie zasobów.

* Zwiększenie limitu żądań. Jeśli stale przekraczasz limity przydziału, rozważ złożenie wniosku o zwiększenie za pośrednictwem portalu Azure lub kontaktując się z pomocą techniczną firmy Microsoft lub dostawcą chmury w przypadku użytkowników spoza platformy Azure.

* Równomiernie rozłóż żądania. Aby uniknąć przekroczenia limitu liczby żądań na minutę (RPM), rozłóż żądania równomiernie w czasie. Wielu dostawców chmury, w tym Azure OpenAI, ocenia liczbę żądań przychodzących w krótkim okresie i może ograniczać przepustowość w przypadku przekroczenia limitu RPM.

UWAGA : Dzięki Azure OpenAI możesz połączyć PAYGO i jednostki PTU, aby sprostać swoim obciążeniom. To hybrydowe podejście pozwala wykorzystać elastyczność PAYGO w przypadku zmiennych obciążeń, zachowując jednocześnie niezawodność i spójność jednostek PTU w przypadku obciążeń o stałym natężeniu. W takim przypadku jednostki PTU sprawdzają się w przypadku obciążeń o stabilnych wymaganiach wydajnościowych, ponieważ zapewniają stałą przepustowość, którą rezerwujesz z wyprzedzeniem, zapewniając niskie wahania opóźnień. Ponadto PAYGO doskonale sprawdza się w obsłudze niepewnych obciążeń, których wykorzystanie może się zmieniać. Opłaty są naliczane na podstawie liczby tokenów użytych na minutę, co oznacza, że płacisz więcej, gdy zużycie jest wysokie, i mniej, gdy jest niskie.

Aktywnie zarządzając swoimi limitami i przepustowościami, przedsiębiorstwa mogą zapewnić sobie niezbędną przepustowość dla swoich aplikacji, kontrolując jednocześnie koszty i utrzymując dostępność usług.

Obserwowalność

Obserwowalność w aplikacjach LLM odnosi się do monitorowania, rejestrowania i śledzenia, aby zapewnić, że aplikacja działa zgodnie z przeznaczeniem i naprawia problemy w przypadku ich wystąpienia. Przyjrzyjmy się każdemu z nich bardziej szczegółowo:

* Monitorowanie — Pomiar kluczowych wskaźników wydajności (KPI), takich jak czasy reakcji, przepustowość, wskaźniki błędów i wykorzystanie zasobów. Dane te są niezbędne do poznania stanu aplikacji i podejmowania mądrych decyzji dotyczących skalowania i optymalizacji.

* Rejestrowanie — Szczegółowe logi powinny rejestrować żądania i odpowiedzi, w tym monity wejściowe i dane wyjściowe modelu. Informacje te są bezcenne dla debugowania, zrozumienia zachowania modelu i poprawy doświadczenia użytkownika.

* Śledzenie — Użyj śledzenia, aby śledzić trasę żądań w aplikacji. Jest to szczególnie ważne w przypadku aplikacji o złożonej architekturze lub wielu modelach i usługach. Śledzenie pomaga zlokalizować wąskie gardła i obszary optymalizacji. W poniższych sekcjach pokażemy, jak to wdrożyć, używając MLflow, Traceloop i Prompt flow. Zacznijmy od MLflow. MLFLOW

MLflow to platforma open source, której celem jest zarządzanie cyklem życia uczenia maszynowego (ML), w tym eksperymentowaniem, powtarzalnością i wdrażaniem. Pomaga ona praktykom uprościć pracę z MLflow dzięki narzędziom do śledzenia eksperymentów, pakowania kodu i zarządzania modelami. Główne komponenty MLflow obejmują śledzenie, rejestr modeli oraz serwer do wdrażania modeli, ułatwiając pracę zespołową i innowacje w projektach ML. MLflow zwiększa obserwowalność modeli LLM, udostępniając narzędzia usprawniające proces wdrażania i monitorowania. Oferuje

ujednolicony interfejs do interakcji z różnymi dostawcami LLM, upraszczając integrację i zarządzanie modelami. Niezależność platformy MLflow ułatwia również bezproblemową integrację i wdrażanie na różnych platformach chmurowych, co dodatkowo wspomaga obserwowalność i zarządzanie modelami LLM. Jak pokazano na listingu 11.3, w tym celu używamy MLflow. Ta podstawowa aplikacja do czatu konsolowego korzysta z Azure OpenAI i losowo używa kilku monitów z listy text_inputs. Możemy ustawić liczbę powtórzeń, korzystając z wielu wątków. Wywołując API dokończenia czatu, logujemy różne funkcje, aby pokazać, jak można zastosować MLflow. Wymagamy, aby MLflow i Prometheus (https://prometheus.io) były zainstalowane i uruchomione w punkcie końcowym, aby to uruchomić. W naszym przypadku uruchamiamy to lokalnie w kontenerze Dockera udostępnionym na porcie 5000. Plik docker-compose jest pokazany na poniższym listingu.

Listing 3: plik docker-compose dla MLflow

```
services:

mlflow:

image: ghcr.io/mlflow/mlflow:latest

command: mlflow server --backend-store-uri /mlflow/mlruns

➥--default-artifact-root /mlflow/artifacts --host 0.0.0.0

ports:

- "5000:5000"

volumes:

- ./mlflow/mlruns:/mlflow/mlruns

- ./mlflow/artifacts:/mlflow/artifacts

prometheus:

image: prom/prometheus:latest

command: --config.file=/etc/prometheus/prometheus.yml

ports:

- "9090:9090"

volumes:

- ./prometheus.yml:/etc/prometheus/prometheus.yml

- ./prometheus/data:/prometheus/data

depends_on:

- mlflow
```

Zaczynamy od uruchomienia kontenera Docker za pomocą polecenia docker compose, jak pokazano: docker compose up -d. Parametr -d uruchamia go jako odłączony, co może być pomocne i działać w tle. Jak opisano na listingu 4, zaczynamy od określenia identyfikatora URI śledzenia MLflow (http://localhost:5000); jest to lokalizacja, w której MLflow będzie przechowywać rejestrowane przez nas dane, a także nada nazwę eksperymentowi (GenAI_book), abyśmy mogli odróżnić go od innych.

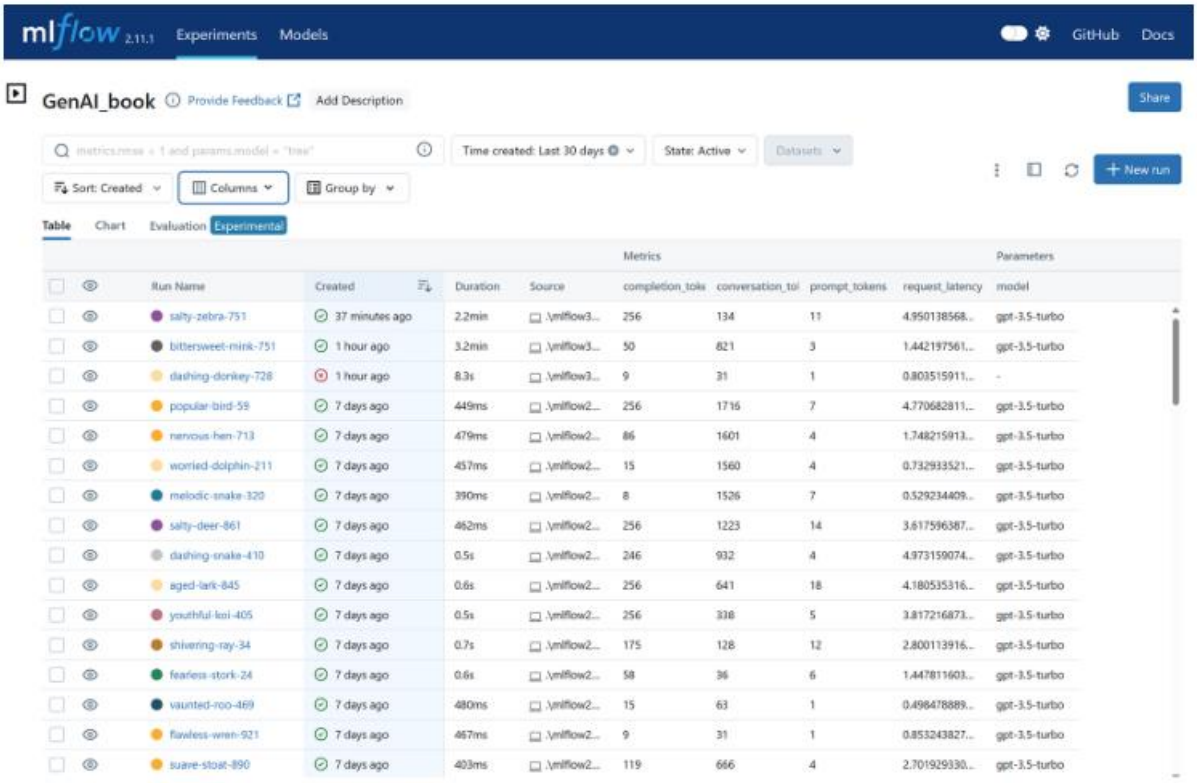
Oczywiście jesteśmy jedynymi użytkownikami tego przykładu, ponieważ działa on lokalnie. Dodatkowo, aby to działało, musimy zainstalować następujące dwie zależności: mlflow i colorama. W przypadku conda można to zainstalować za pomocą conda install -c conda-forge mlflow colorama lub za pomocą pip za pomocą pip install mlflow colorama. Mierzymy takie funkcje, jak liczba tokenów, monity, konwersacje itp. Obliczamy również czas potrzebny na otrzymanie odpowiedzi i jej zapisanie. Do przechowywania wszystkich tych metryk używamy funkcji mlflow.log_metrics(). Parametry używane w żądaniu API przechowujemy również za pomocą funkcji mlflow.log_params().

Listing 4: Przykład obserwowalności MLflow

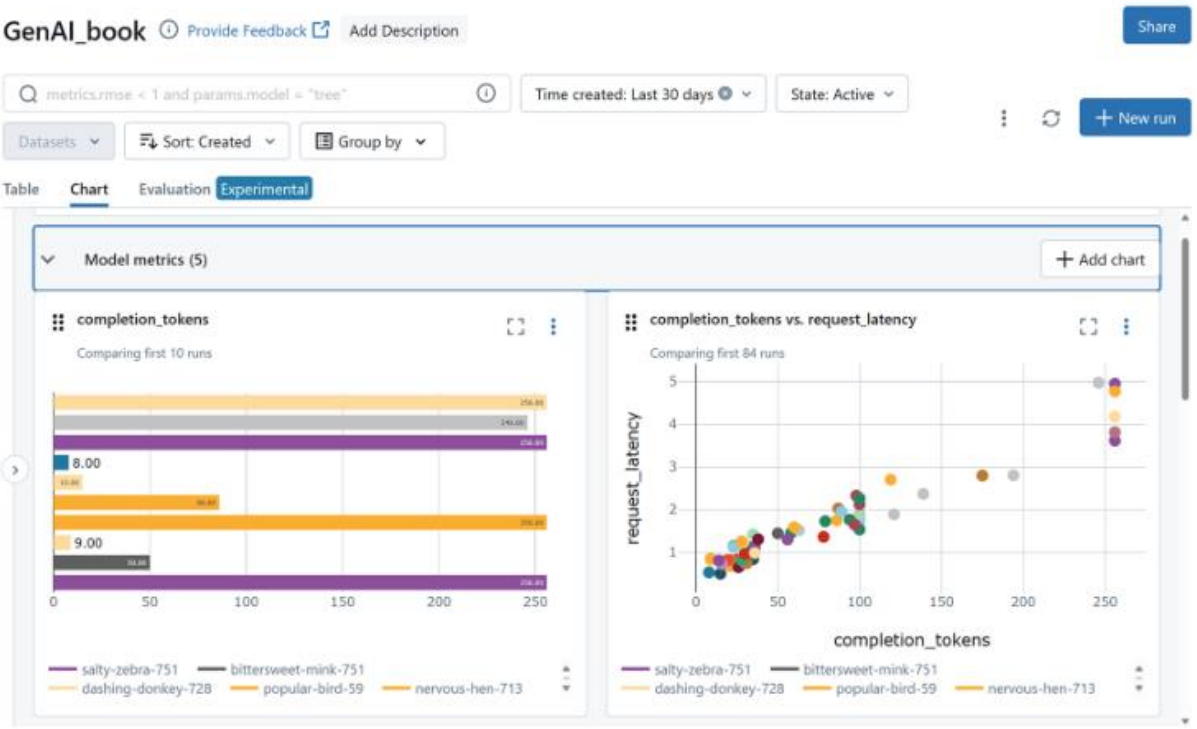
```
import prometheus_client as prom

import mlflow

...

# Set OpenAI API key

API_KEY = os.getenv("OPENAI_API_BOOK_KEY")

MODEL = "gpt-3.5-turbo"

MLFLOW_URI = "http://localhost:5000"

...

# Initialize OpenAI client

client = OpenAI(api_key=API_KEY)

# Set MLflow tracking URI

mlflow.set_tracking_uri(MLFLOW_URI)

mlflow.set_experiment("GenAI_book")

def generate_text(conversation, max_tokens=100)->str:

start_time = time.time()

response = client.chat.completions.create(

model=MODEL,

messages=conversation,

)

latency = time.time() - start_time

message_response = response.choices[0].message.content

# Count tokens in the prompt, and the completion

prompt_tokens = count_tokens(conversation[-1]['content'])

conversation_tokens = count_tokens(str(conversation))

completion_tokens = count_tokens(message_response)
```

```
# Log metrics using MLflow
with mlflow.start_run():
mlflow.log_metrics({
"request_count": 1,
"request_latency": latency,
"prompt_tokens": prompt_tokens,
"completion_tokens": completion_tokens,
"conversation_tokens": conversation_tokens
})
mlflow.log_params({
"model": MODEL,
"temperature": TEMPERATURE,
"top_p": TOP_P,
"frequency_penalty": FREQUENCY_PENALTY,
"presence_penalty": PRESENCE_PENALTY
})
return message_response
if __name__ == "__main__":
conversation = [{"role": "system", "content":
➥"You are a helpful assistant."}]
while True:
user_input = input(f"You: ")
conversation.append({"role": "user", "content": user_input})
output = generate_text(conversation, 256)
print_ai_output(output)
conversation.append({"role": "assistant", "content": output})
```

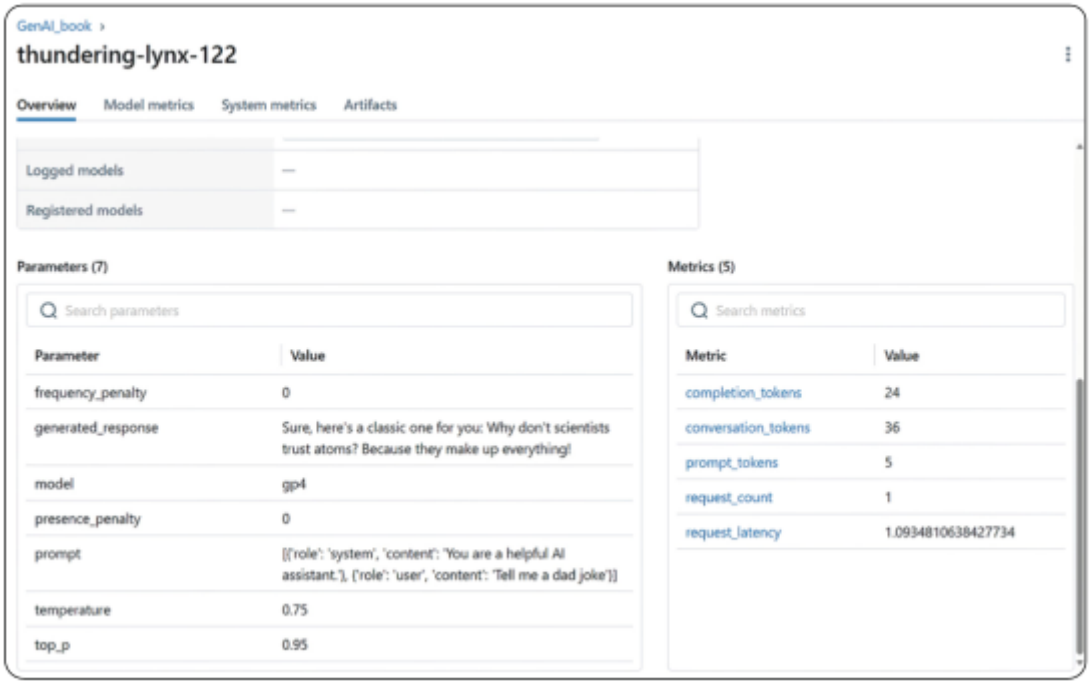
Rejestrowanie tych danych pozwala na porównywanie różnych przebiegów, badanie wydajności modelu i sprawdzanie, jak zmiany parametrów wpływają na wyniki za pomocą interfejsu użytkownika MLflow. Rysunek 6 przedstawia przykład informacji uzyskanych po przeprowadzeniu wielu eksperymentów i umożliwia ich porównanie.



Na rysunku 7 przedstawiono niektóre monitorowane przez nas metryki: completion_tokens i ich związek z opóźnieniem żądania w przypadku kreślenia request_latency.



Rysunek 8 ilustruje, jak możemy również rejestrować niektóre szczegóły komunikatu i wygenerowaną odpowiedź, co jest bardzo przydatne do obserwacji. Oczywiście należy to robić ostrożnie, biorąc pod uwagę prywatność i implikacje prawne dotyczące tego, kto może uzyskać dostęp do tych danych telemetrycznych.



TRACELOOP I OPENLLMETRY

Traceloop to narzędzie do obserwacji aplikacji LLM. Oferuje funkcje takie jak alerty w czasie rzeczywistym i śledzenie wykonania, aby zapewnić wysoką jakość wdrożenia. OpenLLMetry, oparte na OpenTelemetry, to rozszerzenie open source firmy Traceloop, które rozszerza obserwowalność LLM. Integruje się z narzędziami Traceloop i dodaje funkcje monitorowania specyficzne dla LLM, ułatwiając programistom pracę z obserwowalnością LLM, jednocześnie dostosowując się do standardów OpenTelemetry. OpenLLMetry rozszerza funkcjonalność OpenTelemetry o operacje ogólne, takie jak interakcje z bazą danych i API, oraz niestandardowe rozszerzenia dla operacji specyficznych dla LLM. Obejmuje to wywołania dostawców LLM, takich jak OpenAI lub Anthropic, oraz interakcje z bazami danych wektorowych, takimi jak Chroma lub Pinecone. Innymi słowy, OpenLLMetry oferuje specjalistyczny zestaw narzędzi dla aplikacji LLM, ułatwiając deweloperom rozpoczęcie pracy z obserwacją w tej dziedzinie, jednocześnie generując standardowe dane OpenTelemetry, które mogą być kompatybilne z istniejącymi stosami obserwacyjnymi. Zintegrowanie tego z istniejącą aplikacją jest dość proste. Musimy zainstalować pakiet Traceloop SDK (pip install traceloop-sdk). Następnie tworzymy login i pobieramy klucz API na stronie https://app.traceloop.com/. Inicjalizacja jest prosta za pomocą funkcji Traceloop.init(), która automatycznie ją instrumentuje.

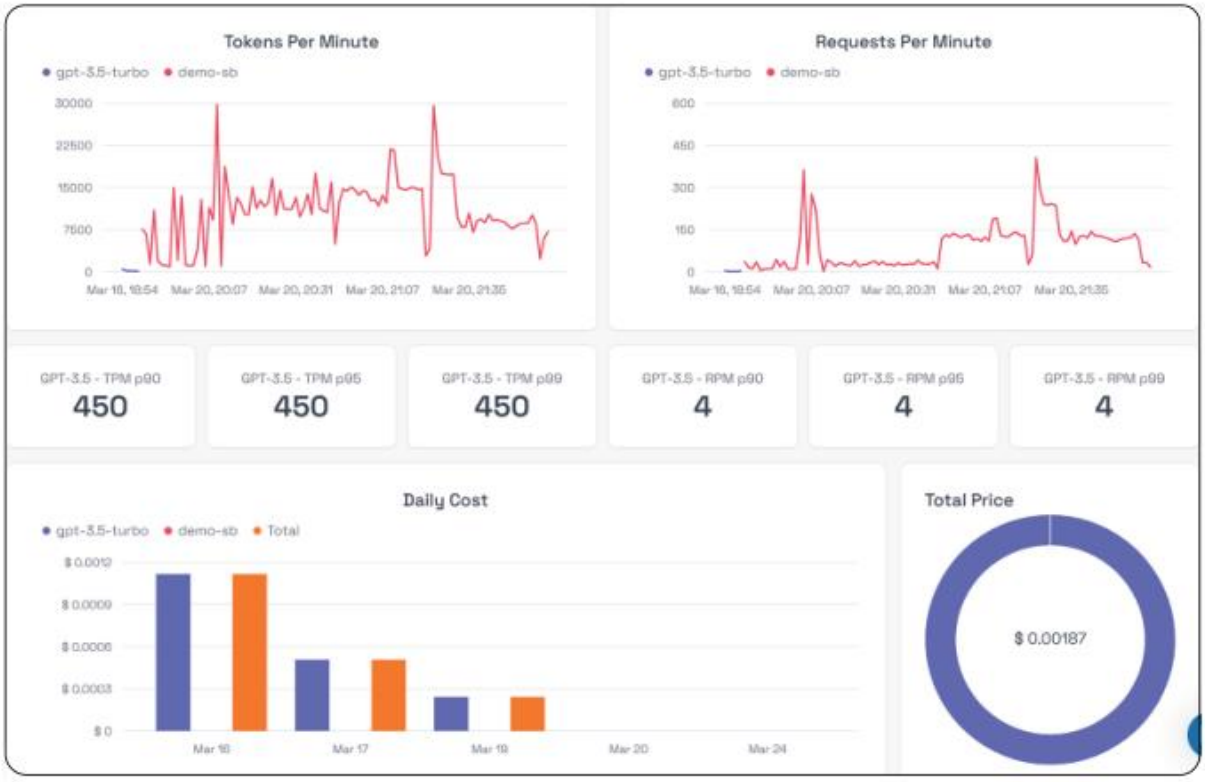
Listing 5: Korzystanie z Traceloop

```
import os

from traceloop.sdk import Traceloop

...

LOAD_TEST_ITERATIONS = 50

# Set OpenAI

API_KEY = os.getenv("AOAI_PTU_KEY")

ENDPOINT = os.getenv("AOAI_PTU_ENDPOINT")
```

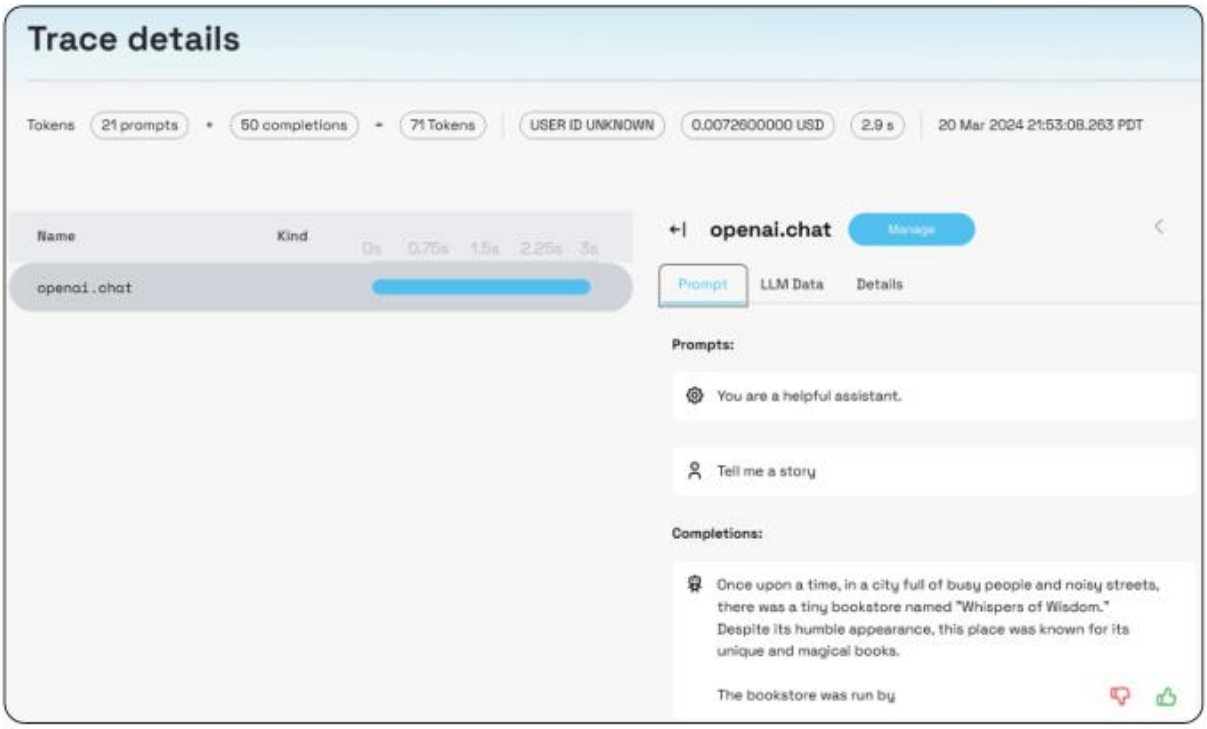
```
...
# Initialize Traceloop
TRACELOOP_API_KEY = os.getenv("TRACELOOP_API_KEY")
Traceloop.init(api_key=TRACELOOP_API_KEY)
client = AzureOpenAI(
azure_endpoint = ENDPOINT,
api_key=API_KEY,
api_version="2024-02-15-preview"
)
# Define the conversation as a list of messages
conversation = [
{"role": "system", "content": "You are a helpful assistant."},
]
# Define a list of test inputs
test_inputs = ["Hello", "How are you?", "What's the weather like?",
➡"Tell me a joke", "Tell me a story", "What's your favorite movie?",
➡"What's the meaning of life?", "What's the capital of France?",
➡"What's the square root of 144?", "What's the largest mammal?"]
print("Starting load test...")
for _ in tqdm(range(LOAD_TEST_ITERATIONS)):
# Generate a random user input
user_input = random.choice(test_inputs)
# Add user input to the conversation
conversation.append({"role": "user", "content": user_input})
# Make the API call
response = client.chat.completions.create(
model=MODEL,
messages=conversation,
temperature=TEMPERATURE,
max_tokens=MAX_TOKENS,
)
```

print("Load test complete.")

Traceloop oferuje również wiele punktów integracji z innymi systemami i różnymi interfejsami API LLM. W naszym przykładzie użyjemy domyślnego pulpitu nawigacyjnego, jak pokazano na rysunku 9.



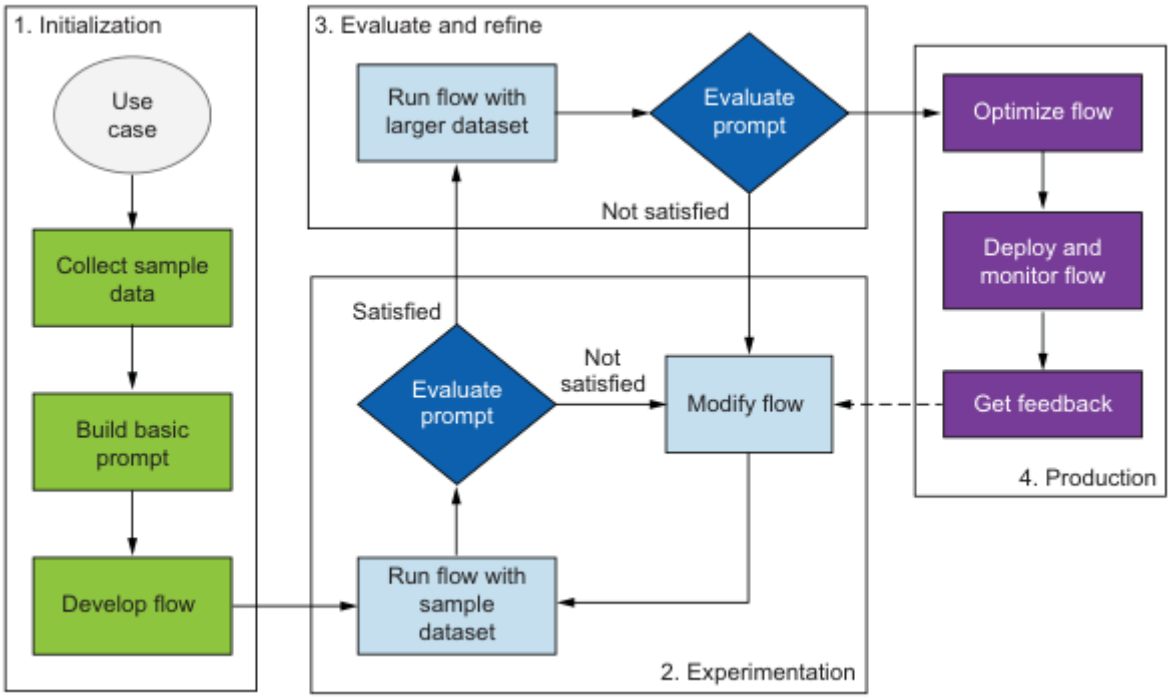
Biorąc pod uwagę możliwość analizy różnych śladów z perspektywy obserwowalności, uzyskujemy wiele szczegółów dotyczących wywołań API (rysunek 10).



W tym przykładzie widzimy monity systemowe, monity użytkownika, uzupełnianie i inne instrumenty, takie jak użycie tokenów. Może to być bardzo przydatna funkcja dla wielu aplikacji korporacyjnych.

PROMPT FLOW

Prompt flow to zestaw narzędzi i funkcji open source firmy Microsoft. Usprawnia on proces tworzenia aplikacji AI, zwłaszcza tych korzystających z modeli LLM. Pomaga w projektowaniu, ocenie i wdrażaniu aplikacji AI, zapewniając programistom prosty interfejs do pracy z modelami LLM. Przepływ monitu to kluczowa funkcja dla programistów, którzy chcą wykorzystywać modele LLM w aplikacjach korporacyjnych, ponieważ ułatwia ona obserwację i wspiera aspekty LLMOps. Umożliwia ona programistom tworzenie wykonywalnych przepływów pracy, które łączą modele LLM, monity i narzędzia Python. Pozwala to programistom łatwiej znajdować i naprawiać błędy oraz ulepszać przepływy, a dodatkowo oferuje funkcje współpracy zespołowej. Programiści mogą tworzyć różne opcje monitu, oceniać ich skuteczność i dostosowywać wydajność modelu LLM w razie potrzeby. Przepływ monitu składa się z czterech etapów, jak pokazano na rysunku 11.



Pierwszy etap, inicjalizacja, obejmuje wybór przypadku biznesowego, zebranie mniejszego zbioru danych i zbudowanie podstawowego monitu i przepływu. Następnie, etap eksperymentowania, wymaga testowania i modyfikowania początkowego monitu, aż do uzyskania zadowalającego rezultatu. Trzeci etap, ocena i udoskonalanie, obejmuje pomiar jakości monitu i wydajności przepływu na większym zbiorze danych, z wprowadzeniem kolejnych korekt i ulepszeń w celu osiągnięcia pożądanego wyniku. Etap produkcyjny obejmuje uruchomienie przepływu do użytku produkcyjnego, śledzenie wykorzystania, informacji zwrotnych i wszelkich problemów, które mogą wystąpić w środowisku produkcyjnym. Szybki przepływ oferuje wiele korzyści podczas przenoszenia aplikacji z fazy rozwoju do produkcji. Pomaga aplikacji sprawnie współpracować z istniejącymi procesami CI/CD oraz zapewnia zaawansowaną kontrolę wersji i narzędzia do współpracy w celu skalowania aplikacji LLM. To kompletne środowisko pozwala programistom wdrażać aplikacje oparte na LLM z większą pewnością, wspierane przez możliwość śledzenia i zrozumienia zachowania modelu w środowisku produkcyjnym. Dlatego szybki przepływ jest kluczowym elementem strategii wdrażania, zapewniając,

że aplikacje korzystające z LLM są wydajne, niezawodne i przygotowane do potrzeb produkcyjnych na poziomie przedsiębiorstwa.

UWAGA : Obsługa modeli polega na wdrażaniu wytrenowanych modeli w celu tworzenia prognoz na podstawie nowych danych. Jest to kluczowy element responsywności i skalowalności aplikacji. Wymaga to jednak znacznych inwestycji w umiejętności, zasoby obliczeniowe w centrach danych, koszty operacyjne oraz specjalistyczny sprzęt, taki jak procesory graficzne z łącznością InfiniBand. Biblioteka oprogramowania open source, taka jak vLLM, mogłaby przynieść korzyści organizacjom rozważającym wdrożenie modelu obsługi.

Wydajny model opiera się na skalowalnej infrastrukturze, która może dostosowywać zasoby do zapotrzebowania, zapewniając dostępność i efektywność kosztową. Strategie buforowania i równoważenie obciążenia są kluczowe dla zmniejszenia opóźnień i równomiernego rozłożenia żądań. Solidna strategia aktualizacji, wykorzystująca wdrożenia blue-green, zapewnia płynne przejścia między modelami z minimalnym czasem przestoju.

Kwestie bezpieczeństwa i zgodności

Bezpieczeństwo i zgodność są kluczowe, szczególnie w przypadku danych użytkowników i potencjalnie wrażliwych informacji. Przestrzeganie najlepszych praktyk pomaga chronić użytkowników i zapewnia zgodność aplikacji z odpowiednimi przepisami i regulacjami.

* Szyfrowanie danych — Szyfruj wrażliwe dane w spoczynku i w trakcie przesyłania, aby chronić je przed nieautoryzowanym dostępem. Używaj bezpiecznych protokołów, takich jak TLS, dla danych w trakcie przesyłania i korzystaj z funkcji szyfrowania oferowanych przez dostawcę chmury dla danych w spoczynku.

* Kontrola dostępu — Wdrażaj ścisłe kontrole dostępu, aby zapewnić dostęp do danych produkcyjnych i infrastruktury tylko upoważnionym osobom. Stosuj kontrolę dostępu opartą na rolach (RBAC) i zasadę najmniejszych uprawnień (PoLP), aby zminimalizować ryzyko naruszenia bezpieczeństwa danych.

* Audyty zgodności — Regularnie audytuj swoją aplikację i jej infrastrukturę pod kątem zgodności z odpowiednimi przepisami i normami, takimi jak RODO, HIPAA lub CCPA, w zależności od domeny i zasięgu geograficznego aplikacji. Może to obejmować przeprowadzanie ocen bezpieczeństwa, skanowanie podatności i sprawdzanie zgodności.

* Wykrywanie anomalii – wdrażanie systemów wykrywania anomalii w celu monitorowania nietypowych działań, które mogą wskazywać na naruszenie bezpieczeństwa lub niewłaściwe użycie systemu. Obejmuje to monitorowanie nietypowych wzorców użytkowania lub nieautoryzowanych prób dostępu, umożliwiając szybką reakcję na potencjalne zagrożenia.

Usługa Azure OpenAI oferuje wiele z tych funkcji standardowo, aby sprostać wymaganiom przedsiębiorstw w zakresie gotowości i zgodności. Zgodnie z zapotrzebowaniem większości przedsiębiorstw, inni dostawcy chmury, tacy jak AWS i GCP, oferują pewne wersje tych mechanizmów kontroli.

Zagadnienia operacyjne GenAI

Aspekty operacyjne aplikacji GenAI, zwłaszcza tych wykorzystujących LLM, takich jak GPT-4, mają kluczowe znaczenie dla zapewnienia płynnego i wydajnego funkcjonowania tych systemów. Zrozumienie i zarządzanie kluczowymi wskaźnikami operacyjnymi, takimi jak tokeny, opóźnienia, liczba żądań na sekundę (RPS) i czas do pierwszego bajtu (TTFB), ma kluczowe znaczenie dla optymalizacji

wydajności, doświadczenia użytkownika i kosztów. Przyjrzyjmy się definicji, lepiej zrozumiemy znaczenie tych kryteriów operacyjnych i omówimy, jak skutecznie je mierzyć i nimi zarządzać.

Zagadnienia dotyczące niezawodności i wydajności

Każdy system produkcyjny, w tym aplikacja GenAI, musi być niezawodny i wydajny, aby sprostać potrzebom i oczekiwaniom użytkowników. Oznacza to, że system powinien być w stanie poradzić sobie z różnymi awariami i scenariuszami. System zarządzania API lub system proxy może pomóc w wielu z tych aspektów, które omówimy poniżej:

* Narzędzia monitorujące — korzystaj z narzędzi i usług monitorujących, aby stale mierzyć te wskaźniki operacyjne. Narzędzia takie jak Prometheus (https://prometheus.io) do gromadzenia metryk i Grafana (https://grafana.com) do wizualizacji mogą zapewnić wgląd w czasie rzeczywistym w wydajność aplikacji. Dostawcy usług w chmurze oferują również natywne rozwiązania do monitorowania, które można wykorzystać.

* Testowanie wydajności — Regularnie przeprowadzaj testy wydajności, aby symulować różne warunki obciążenia i mierzyć reakcję aplikacji. Narzędzia takie jak Apache JMeter (https://jmeter.apache.org) lub Locust (https://locust.io) mogą symulować interakcję wielu użytkowników z aplikacją, aby ocenić jej przepustowość i opóźnienia w warunkach dużego obciążenia.

* Techniki optymalizacji — Wdrożenie skutecznych technik optymalizacji ma kluczowe znaczenie dla ogólnej wydajności aplikacji, wykorzystania zasobów i doświadczenia użytkownika:

– Zarządzanie tokenami — Optymalizacja wykorzystania tokenów poprzez udoskonalenie monitów i odpowiedzi wejściowych. Może to obejmować usuwanie zbędnego tekstu, stosowanie bardziej efektywnych technik kodowania lub dostosowywanie modelu w celu generowania krótszych, bardziej zwięzłych wyników bez utraty jakości.

– Buforowanie — Wdrażaj strategie buforowania dla często żądanych informacji, aby zmniejszyć opóźnienia i obciążenie obliczeniowe systemu. Jest to szczególnie skuteczne w przypadku danych statycznych lub rzadko zmieniających się.

– Równoważenie obciążenia i automatyczne skalowanie — Używaj modułów równoważenia obciążenia, aby równomiernie rozłożyć ruch w całej infrastrukturze i wdróż automatyczne skalowanie, aby dynamicznie dostosowywać zasoby w zależności od zapotrzebowania. Pomaga to utrzymać niskie opóźnienia i wysoki wskaźnik RPS, zapewniając, że system może obsługiwać skoki ruchu bez ręcznej interwencji.

* Zarządzanie kosztami — Monitoruj i zarządzaj kosztami związanymi z metrykami operacyjnymi, zwłaszcza wykorzystaniem tokenów, ponieważ ma to bezpośredni wpływ na koszt korzystania z interfejsów API LLM. W razie potrzeby wdrażaj systemy kwot lub limity przepustowości, aby zapobiec nieoczekiwanym skokom obciążenia.

Koncentrując się na tych aspektach operacyjnych oraz stale monitorując i optymalizując je w oparciu o dane rzeczywiste, programiści mogą zapewnić, że ich aplikacje GenAI są funkcjonalne, a jednocześnie wydajne, skalowalne i ekonomiczne. To holistyczne podejście do zarządzania operacyjnego ma kluczowe znaczenie dla sukcesu każdej aplikacji wykorzystującej potencjał LLM.

Tożsamości zarządzane

Platforma Azure OpenAI ma kluczową przewagę nad OpenAI lub innymi dostawcami LLM w zakresie wykorzystania zarządzanych tożsamości do uwierzytelniania. Ta metoda jest zgodna z najlepszymi

praktykami wdrożeń produkcyjnych w przedsiębiorstwach, zwiększając bezpieczeństwo i ułatwiając zarządzanie poświadczeniami. Tożsamości zarządzane eliminują konieczność bezpośredniego przetwarzania kluczy, zmniejszając ryzyko ich ujawnienia i upraszczając proces zmiany poświadczeń. Oferują one również zautomatyzowany sposób uwierzytelniania usług działających na platformie Azure z innymi zasobami platformy Azure, wykorzystując usługę Azure Active Directory (AAD) do zarządzania tożsamościami (znaną również jako Entra ID). Korzystając z zarządzanych tożsamości w usłudze Azure OpenAI, przedsiębiorstwa mają do dyspozycji kilka metod uwierzytelniania — RBAC i Entra ID. Pierwszy z nich pozwala na realizację bardziej złożonych scenariuszy bezpieczeństwa i obejmuje przypisywanie ról (np. użytkownika lub współautora), aby umożliwić wywołania API bez uwierzytelniania opartego na kluczu. Drugi natomiast służy do uwierzytelniania naszego zasobu OpenAI za pomocą tokenu nośnika uzyskanego za pośrednictwem interfejsu wiersza poleceń platformy Azure. Wymaga on niestandardowej nazwy subdomeny i jest odpowiedni dla aplikacji działających w usługach platformy Azure, takich jak maszyny wirtualne, aplikacje funkcji i zestawy skalowania maszyn wirtualnych. Tożsamości zarządzane oferują szereg korzyści w porównaniu z tradycyjnymi metodami uwierzytelniania opartymi na kluczach, zwłaszcza w zakresie bezpieczeństwa i zarządzania. Do najważniejszych zalet należą:

* Brak konieczności zarządzania poświadczeniami — Tożsamości zarządzane eliminują konieczność zarządzania przez deweloperów sekretami, poświadczeniami, certyfikatami i kluczami używanymi do zabezpieczania komunikacji między usługami.

* Automatyczna rotacja poświadczeń — Przypisane przez system tożsamości zarządzane są powiązane z cyklem życia zasobu platformy Azure, a platforma Azure automatycznie obsługuje cykl życia poświadczeń, w tym ich rotację.

* Zwiększone bezpieczeństwo — Ponieważ poświadczenia nie są przechowywane w kodzie, ryzyko wycieku poświadczeń jest mniejsze. Tożsamości zarządzane wykorzystują również usługę AAD do uwierzytelniania, co jest bezpieczniejsze niż przechowywanie i zarządzanie kluczami w aplikacji.

* Uproszczone zarządzanie dostępem — Tożsamości zarządzane mogą uzyskać dostęp do innych zasobów platformy Azure obsługujących uwierzytelnianie usługi Azure AD, co upraszcza zarządzanie dostępem. Ponadto, przypisane przez użytkownika tożsamości zarządzane mogą być używane przez wiele zasobów, co może być szczególnie przydatne w złożonych środowiskach i aplikacjach wymagających skalowania.

Te korzyści przyczyniają się do bezpieczniejszego i wydajniejszego środowiska zarządzania dostępem do zasobów platformy Azure, dzięki czemu zarządzane tożsamości są preferowanym rozwiązaniem w wielu scenariuszach korporacyjnych. Poniższa lista przedstawia prosty przykład implementacji zarządzanej tożsamości przy użyciu Azure OpenAI. Należy pamiętać, że może to wymagać zainstalowania pakietu Azure Identity, co można zrobić za pomocą polecenia pip: pip install azure-identity.

Listing 6: Korzystanie z zarządzanych tożsamości z usługą Azure OpenAI

```
import os

from openai import AzureOpenAI

from azure.identity import DefaultAzureCredential,

➥get_bearer_token_provider

AZURE_ENDPOINT = os.getenv("AOAI_ENDPOINT")
```

```
API_VERSION = "2024-02-15-preview"

token_provider = get_bearer_token_provider(

DefaultAzureCredential(),

"https://cognitiveservices.azure.com/.default"

)

client = AzureOpenAI(

api_version=API_VERSION,

azure_endpoint=AZURE_ENDPOINT,

azure_ad_token_provider=token_provider,
```

Buforowanie

Wdrożenie buforowania podczas korzystania z platformy LLM firmy OpenAI w aplikacji produkcyjnej to strategiczne posunięcie mające na celu poprawę wydajności i obniżenie kosztów. Buforowanie przechowuje często żądane dane w systemie pamięci masowej o szybszym dostępie, co pozwala na zmniejszenie opóźnień, ponieważ powtarzające się zapytania mogą być obsługiwane szybko. Poprawia to komfort użytkowania i minimalizuje koszty operacyjne poprzez redukcję liczby niezbędnych wywołań API, często związanych z opłatami. Co więcej, usługi zazwyczaj nakładają limity przepustowości, aby zapobiec nadmiernemu obciążeniu, a buforowanie pomaga nam przestrzegać tych limitów, jednocześnie zapewniając responsywność usługi. Jeśli chodzi o najlepsze praktyki buforowania w Redis, kluczowe jest projektowanie kluczy pamięci podręcznej w sposób unikatowy reprezentujących każde żądanie i jego kontekst. Skuteczna strategia unieważniania, taka jak ustawienie czasu życia (TTL) dla kluczy, zapewnia, że pamięć podręczna nie będzie dostarczać nieaktualnych informacji. Wzorzec „cache-aside" to zalecane podejście, w którym aplikacja najpierw sprawdza pamięć podręczną, a w przypadku braku odpowiedzi pobiera dane ze źródła, aktualizuje pamięć podręczną, a następnie zwraca odpowiedź. Monitorowanie wskaźników trafień i metryk wydajności pamięci podręcznej jest niezbędne do oceny jej skuteczności i wprowadzenia niezbędnych optymalizacji. Ważne jest, aby płynnie obsługiwać braki w pamięci podręcznej i zapewnić poprawne działanie aplikacji nawet w przypadku tymczasowej niedostępności. Możemy zilustrować, jak buforowanie generacji LLM może przynieść aplikacji znaczne korzyści pod względem kosztów i wydajności. Nie powinniśmy jednak buforować niczego bez wyraźnego powodu, licząc na poprawę sytuacji, ale rozważyć to w kontekście przypadku użycia i powiązanych typów generacji. W naszym przykładzie buforowania użyjemy Redis i oprzemy się na implementacji RAG opisanej wcześniej w rozdziale 8. Korzystając z tego samego kontenera Docker, wykorzystamy bibliotekę RedisVL, bibliotekę Pythona przeznaczoną do zadań takich jak wyszukiwanie semantyczne i potoki RAG w czasie rzeczywistym. Zapewnia ona łatwy w użyciu interfejs do wyszukiwania wektorowego i zarządzania indeksami. RedisVL jest oparty na kliencie redis-py i pomaga zintegrować możliwości Redis z aplikacjami opartymi na sztucznej inteligencji. Zaczynamy od instalacji za pomocą pip: pip install redisvl. Kontynuujemy, wyświetlając listę wszystkich indeksów w bazie danych Redis, która ma tylko jeden indeks, posts, z naszej implementacji RAG wcześniej w rozdziale 8. Polecenie rvl index listall umożliwiające wyświetlenie wszystkich indeksów wygląda następująco:

```
11:33:52 [RedisVL] INFO Indices:

11:33:52 [RedisVL] INFO 1. Posts
```

Następnie inicjujemy pamięć podręczną, która jest tworzona, jeśli pamięć podręczna nie istnieje. Inicjalizacja pamięci podręcznej wymaga kilku parametrów — nazwy (z uwzględnieniem wielkości liter), prefiksu wpisów skrótu, ciągu połączenia (lokalnego w naszym przypadku, ponieważ uruchamiamy go lokalnie w Dockerze) oraz progu odległości. Próg odległości może się różnić w zależności od kodu osadzania i przypadku użycia i można go zmieniać na bieżąco.

Nasza funkcja, answer_question(), przyjmuje pytanie i używa metody check() na instancji llmcache do przeszukania pytania w pamięci podręcznej. Jeśli pamięć podręczna zawiera wyniki, zwraca odpowiedź. Jeśli pamięć podręczna jest pusta, wywołuje funkcję generate_response, aby pobrać odpowiedź od klienta OpenAI, która jest następnie przechowywana w pamięci podręcznej. Należy zauważyć, że część kodu została pominięta dla uproszczenia. Poniższy listing pokazuje całość.

Listing 7: Korzystanie z pamięci podręcznej Redis do odpowiedzi OpenAI

```
from openai import AzureOpenAI
from redisvl.extensions.llmcache import SemanticCache
import numpy as np
...
# Set your OpenAI API key
AOAI_API_KEY = os.getenv("AOAI_KEY")
...

def initialize_cache():
    # Initialize the semantic cache
    llmcache = SemanticCache(
        name="GenAIBookCache",
        prefix="bookcache",
        redis_url="redis://localhost:6379",
        distance_threshold=0.1
    )
    return llmcache

# Define a list of questions
input_questions = ["What is the capital of UK?", ...
                   "What is the capital of Japan?"]

def generate_response(conversation, max_tokens=25)->str:
    response = client.chat.completions.create(
        ...
    )
    return response.choices[0].message.content

def answer_question(question: str) -> str:
    conversation = [{"role": "assistant", "content": question}]

    results = llmcache.check(prompt=question)
    if results:
        answer = results[0]["response"]
    else:
        answer = generate_response(conversation)
        llmcache.store(prompt=question, response=answer)
    return answer

if __name__ == "__main__":
    llmcache = initialize_cache()

    times_without_cache = []
    times_with_cache = []

    for question in input_questions:
        # Without caching
        start_time = time.time()
        answer = generate_response([{"role": "assistant",
```

Index name

Redis key prefix for hash entries

Redis connection url string

Semantic cache distance threshold

```
            ➥"content": question}])
    end_time = time.time()
    times_without_cache.append(end_time-start_time)

    # With caching
    start_time = time.time()
    answer = answer_question(question)
    end_time = time.time()
    times_with_cache.append(end_time-start_time)

avg_time_without_cache = np.mean(times_without_cache)
avg_time_with_cache = np.mean(times_with_cache)

print(f"Avg time taken without cache: {avg_time_without_cache}")
print(f"Avg time taken with LLM cache enabled: {avg_time_with_cache}")
print(f"Percentage of time saved: {round((avg_time_without_cache -
        ➥avg_time_with_cache) / avg_time_without_cache * 100, 2)}%")
```

Po uruchomieniu tego przykładowy wynik to

11:16:17 redisvl.index.index INFO Index already exists, not overwriting.

Cache hit for prompt: What is the capital of UK?, answer: London

...

Cache miss for prompt: What is the capital of India?, added to

➥cache with response: The capital of India is New Delhi.

Avg time taken without cache: 0.7652951717376709

Avg time taken with LLM cache enabled: 0.23438820838928223

Percentage of time saved: 69.37%

Mechanizm TTL określa, jak długo dany element danych powinien być przechowywany w pamięci podręcznej, zanim zostanie uznany za nieaktualny i będzie można go usunąć. W Redis, po upływie TTL, dane z pamięci podręcznej są automatycznie usuwane, co zapobiega wyświetlaniu użytkownikom nieaktualnych informacji. Pomaga to utrzymać aktualność danych, do których aplikacja uzyskuje dostęp. Można to ustawić w następujący sposób: llmcache.set_ttl(5) # 5 sekund. Możemy użyć polecenia rvl stats z nazwą pamięci podręcznej jako argumentem, aby wyświetlić szczegóły pamięci podręcznej. Rysunek 11.12 przedstawia wynik tego polecenia: rvl stats —i GenAIBookCache.

Statistics:

Stat Key	Value
num_docs	10
num_terms	78
max_doc_id	10
num_records	113
percent_indexed	1
hash_indexing_failures	0
number_of_uses	46
bytes_per_record_avg	5.69912
doc_table_size_mb	0.00135422
inverted_sz_mb	0.000614166
key_table_size_mb	0.000359535
offset_bits_per_record_avg	8
offset_vectors_sz_mb	0.000123024
offsets_per_term_avg	1.14159
records_per_doc_avg	11.3
sortable_values_size_mb	0
total_indexing_time	4.008
total_inverted_index_blocks	33033
vector_index_sz_mb	3.00867

Przedstawiliśmy komponenty, które należy uwzględnić, aby aplikacja GenAI była skalowalna i operacyjna. Pozostaje jeszcze jeden temat do omówienia: LLMOps i MLOps. Nie służą one tylko do uruchamiania aplikacji AI, ale do robienia tego w sposób łatwy w utrzymaniu, etyczny i skalowalny. Dlatego są uważane za niezbędne dla każdego przedsiębiorstwa, które chce efektywnie wykorzystać technologię AI. Przyjrzyjmy się im bliżej.

LLMOps i MLOps

Operacje uczenia maszynowego (MLOps) stosują zasady i najlepsze praktyki DevOps do tworzenia, wdrażania i zarządzania modelami i aplikacjami ML. MLOps ma na celu usprawnienie cyklu życia ML, od przygotowywania i eksperymentowania danych po trenowanie i obsługę modeli, przy jednoczesnym zapewnieniu jakości, niezawodności i skalowalności. LLMOps to specjalistyczna dziedzina MLOps, która koncentruje się na operacyjnych aspektach LLM. LLM to modele głębokiego uczenia, które mogą generować tekst w języku naturalnym i wykonywać różne zadania przetwarzania języka naturalnego (NLP) w oparciu o dostarczone dane wejściowe. Przykładami LLM są GPT-4, BERT i podobne zaawansowane systemy sztucznej inteligencji. LLMOps wprowadza narzędzia i najlepsze praktyki, które pomagają zarządzać cyklem życia LLM i aplikacji opartych na LLM, takie jak szybkie projektowanie, dostrajanie, wdrażanie, monitorowanie i zarządzanie. LLMOps zajmuje się również wyjątkowymi wyzwaniami i ryzykiem związanym z LLM, takimi jak stronniczość, halucynacje, szybkie wstrzykiwanie danych i kwestie etyczne. Zarówno LLMOps, jak i MLOps mają pewne wspólne cele i wyzwania, takie jak automatyzacja i koordynacja procesu uczenia maszynowego (ML); zapewnienie powtarzalności, identyfikowalności i wersjonowania danych, kodu, modeli i eksperymentów; monitorowanie i optymalizacja wydajności, dostępności i wykorzystania zasobów modeli i aplikacji w środowisku produkcyjnym; wdrażanie środków bezpieczeństwa, ochrony prywatności i zgodności w celu ochrony danych i modeli przed nieautoryzowanym dostępem i niewłaściwym wykorzystaniem; oraz wdrażanie pętli sprzężenia zwrotnego i cykli ciągłego doskonalenia w celu aktualizacji i udoskonalania modeli i aplikacji w oparciu o zmieniające się wymagania i zachowania użytkowników. LLMOps i MLOps mają jednak również pewne wyraźne różnice, a przejście z MLOps na LLMOps to zmiana paradygmatu – szczególnie w zakresie danych, złożoności modelu (w tym rozmiaru) i wyników modelu w kontekście generowania:

* Dane — modele LLM są wstępnie trenowane na ogromnych zbiorach danych tekstowych, takich jak korpus Common Crawl, i mogą być dostosowywane do konkretnych przypadków użycia za pomocą szybkich technik inżynieryjnych i dostrajania. Zmniejsza to potrzebę intensywnego gromadzenia i

etykietowania danych oraz wprowadza ryzyko wycieku i zanieczyszczenia danych z danych wstępnie trenowanych.

* Zasoby obliczeniowe — modele GenAI, takie jak modele LLM, są bardzo duże i złożone, często składają się z miliardów parametrów i wymagają specjalistycznego sprzętu i infrastruktury do trenowania i uruchamiania, takiej jak zaawansowane procesory graficzne, pamięć itp. Stanowi to poważne wyzwanie dla przechowywania, dystrybucji, wnioskowania, kosztów i efektywności energetycznej modelu. Wyzwanie to jest dodatkowo wzmacniane, gdy chcemy skalować do wielu użytkowników, aby obsługiwać przychodzące żądania bez obniżania wydajności.

* Generowanie modeli – LLM-y są projektowane tak, aby generować spójny i kontekstowo poprawny tekst, a nie skupiać się na faktach. Prowadzi to do różnych zagrożeń, takich jak wzmacnianie stronniczości, halucynacje, podstępne wstrzyknięcia i kwestie etyczne. Zagrożenia te wymagają starannej oceny i strategii ograniczania, takich jak odpowiedzialne ramy sztucznej inteligencji, nadzór ludzki i narzędzia wyjaśniające.

Tabela 4 przedstawia najważniejsze różnice w przejściu z MLOps na LLMOps

Obszar: Tradycyjne MLOps: LLMOps

Grupa docelowa: Inżynierowie ML, analitycy danych: Deweloperzy aplikacji, inżynierowie ML i analitycy danych

Komponenty: Model, dane, środowiska wnioskowania, funkcje:

LLM, monity, tokeny, generacje, API, osadzenia, wektorowe bazy danych

Metryki: Dokładność (wynik F1, precyzja, odwołanie itp.); Jakość (podobieństwo), ugruntowanie (dokładność), koszt (tokeny), opóźnienie, oceny (Perplexity, BLEU, ROUGE itp.); Modele: Zazwyczaj budowane od podstaw: Zazwyczaj wstępnie zbudowane z wnioskowaniem za pośrednictwem API i wieloma wersjami w środowisku produkcyjnym jednocześnie.

Obawy etyczne: Błąd w danych szkoleniowych: Nadużycie i generowanie szkodliwych, fałszywych i stronniczych wyników.

DLACZEGO LLMOPS I MLOPS?

LLMOps i MLOps są kluczem do odpowiedzialnego i efektywnego wdrażania modeli LLM i ML, zapewniając standardy etyczne i wydajnościowe. Rozwiązują one problemy takie jak powolny rozwój, niespójna jakość modeli i wysokie koszty, jednocześnie zapewniając korzyści takie jak szybkość, spójność i zarządzanie ryzykiem. LLMOps obejmuje narzędzia i praktyki zarządzania modelami LLM, w tym szybką inżynierię, dostrajanie i zarządzanie, co przekłada się na szybszy rozwój, lepszą jakość, redukcję kosztów i kontrolę ryzyka. Ze względu na ich złożoność, efektywne zarządzanie ma kluczowe znaczenie dla wydajności i opłacalności generatywnych modeli AI. Ważnymi czynnikami w LLMOps są wybór modelu, strategie wdrażania i kontrola wersji. Odpowiedni rozmiar i konfiguracja modelu są niezbędne, w miarę możliwości dostosowane do konkretnych danych. Opcje między usługami w chmurze a infrastrukturą prywatną równoważą wygodę i bezpieczeństwo danych. Wersjonowanie i zautomatyzowane potoki wspierają płynne aktualizacje i wycofywanie zmian, umożliwiając ciągłą integrację i wdrażanie. Wdrożenie LLMOps zapewnia skuteczne i etyczne wykorzystanie generatywnej AI, maksymalizując korzyści i minimalizując ryzyko. LLMOps i MLOps mają kluczowe znaczenie dla wdrażania aplikacji AI w środowisku produkcyjnym. Zapewniają one niezbędną infrastrukturę, gwarantującą, że aplikacje AI są operacyjne, stabilne, odpowiedzialne i skalowalne zgodnie z zapotrzebowaniem użytkowników. Dla programistów i specjalistów technicznych te frameworki

oferują sposób na utrzymanie kontroli jakości, przestrzeganie standardów zgodności i etycznych oraz ekonomiczne zarządzanie aplikacjami AI. W środowisku korporacyjnym, gdzie niezawodność i skalowalność mają kluczowe znaczenie, LLMOps i MLOps są niezbędne do pomyślnej integracji technologii AI.

SYSTEMY MONITOROWANIA I TELEMETRII

Chociaż wydajne systemy LLM są w stanie dostarczać wartościowe rezultaty biznesowe, wymagają starannego monitorowania i zarządzania, aby zapewnić optymalną wydajność, dokładność, bezpieczeństwo i komfort użytkowania. Monitorowanie jest ważnym elementem systemów LLMOps i MLOps, ponieważ pokazuje, jak dobrze modele i aplikacje działają w środowisku produkcyjnym. Ciągły monitoring jest niezbędny dla systemów LLMOps, podobnie jak dla wielu systemów produkcyjnych. Pomaga zespołom LLMOps szybko rozwiązywać problemy, zapewniając szybkość i niezawodność systemu. Monitorowanie obejmuje wskaźniki wydajności, takie jak czas reakcji, przepustowość i wykorzystanie zasobów, umożliwiając szybką interwencję w przypadku opóźnień lub spadków wydajności. Śledzenie danych telemetrycznych ma kluczowe znaczenie w tym procesie, dostarczając cennych informacji na temat zachowania modelu i umożliwiając ciągłe doskonalenie. Ponadto etyczne wdrażanie sztucznej inteligencji musi być wykrywane pod kątem stronniczości lub szkodliwych wyników. Stosując metody monitorowania uwzględniające uczciwość, zespoły LLMOps zapewniają etyczne działanie systemów LLM, minimalizując niepożądane stronniczości i zwiększając zaufanie użytkowników. Częste aktualizacje i konserwacja modelu, wspierane przez zautomatyzowane procesy, gwarantują, że LLM pozostaje na bieżąco z najnowszymi osiągnięciami i trendami danych, gwarantując stałą skuteczność i elastyczność.

Lista kontrolna wdrożenia produkcyjnego

W tym rozdziale omówiliśmy wiele tematów. Zanim zakończymy, podsumujmy niektóre z porad w prostej liście kontrolnej, która może być przydatna jako przewodnik referencyjny podczas wdrażania aplikacji w środowisku produkcyjnym. Poniższe kategorie są takie same, jak te opisane wcześniej w rozdziale. Oczywiście, jak w przypadku większości tych porad, jest to niekompletne i powinno być traktowane jako część szerszego zestawu obowiązków:

* Skalowanie i wdrażanie

– Ocena zasobów obliczeniowych – Określ wymagania sprzętowe i programowe swoich generatywnych modeli AI i upewnij się, że infrastruktura może je efektywnie obsługiwać.

– Jakość i dostępność danych – Wdróż solidne procesy walidacji danych, kontroli jakości i ciągłego monitorowania, aby zapewnić dokładność i trafność danych.

– Wydajność i niezawodność modelu – Skonfiguruj regularne procesy testowania i walidacji, aby monitorować wydajność modeli. Zaplanuj redundancję, przełączanie awaryjne i odzyskiwanie po awarii, aby zapewnić wysoką dostępność.

– Bezpieczeństwo i zgodność – Stosuj szyfrowanie, kontrolę dostępu i regularne audyty zgodności. Upewnij się, że Twoje modele są zgodne z przepisami takimi jak RODO lub HIPAA.

– Zarządzanie kosztami — Ściśle monitoruj i zarządzaj kosztami wdrażania i utrzymania modeli. Bądź przygotowany na kompromisy między kosztami a wydajnością.

– Integracja systemów — Upewnij się, że generatywne modele AI można łatwo zintegrować z istniejącymi systemami i przepływami pracy.

– Uwzględnienie czynnika ludzkiego — Zaprojektuj modele tak, aby w razie potrzeby uwzględniały nadzór i interwencję człowieka.

– Kwestie etyczne — Podczas wdrażania modeli należy uwzględnić implikacje etyczne, takie jak stronniczość i uczciwość.

* Najlepsze praktyki wdrażania produkcyjnego

– Metryki wnioskowania LLM — Skoncentruj się na kluczowych metrykach, takich jak czas do pierwszego tokena (TTFT), czas na token wyjściowy (TPOT), opóźnienie i przepustowość. Użyj narzędzi takich jak MLflow do śledzenia tych metryk.

– Zarządzanie opóźnieniami — Poznaj różne punkty opóźnień i dokładnie je zmierz. Weź pod uwagę wpływ rozmiaru promptu i rozmiaru modelu na opóźnienie.

– Skalowalność — Wykorzystaj modele PTU i PAYGO, aby efektywnie skalować swoją aplikację. Wykorzystaj zarządzanie API do kolejkowania, ograniczania przepustowości i zarządzania limitami wykorzystania.

– Limity i limity przepustowości — wdrażaj strategie efektywnego zarządzania limitami i limitami przepustowości, w tym zrozumienie limitów, monitorowanie wykorzystania i implementację logiki ponawiania prób.

– Obserwowalność — używaj narzędzi takich jak MLflow, Traceloop i Prompt flow do monitorowania, rejestrowania i śledzenia aplikacji w celu poprawy wydajności i komfortu użytkowania.

– Bezpieczeństwo i zgodność — szyfruj dane, kontroluj dostęp, przeprowadzaj audyty zgodności i wdrażaj systemy wykrywania anomalii.

* LLMOps i MLOps

– Wdrażaj frameworki LLMOps i MLOps — upewnij się, że Twoja aplikacja jest zgodna z najlepszymi praktykami LLMOps i MLOps, aby zapewnić etyczne, łatwe w utrzymaniu, etyczne i skalowalne rozwiązania AI.

– Systemy monitorowania i telemetrii — stosuj metody monitorowania uwzględniające uczciwość i śledzenie danych telemetrycznych, aby zapewnić etyczne wdrażanie AI i ciągłe doskonalenie modeli.

Podsumowanie

* Modele generatywnej sztucznej inteligencji są złożone i wymagają dużych zasobów, co wymaga starannego rozważenia jakości danych, wydajności, bezpieczeństwa, kosztów i konsekwencji etycznych.

* W przypadku wdrożeń produkcyjnych musimy przestrzegać kilku najlepszych praktyk: monitorować kluczowe wskaźniki, optymalizować opóźnienia, zapewniać skalowalność, wdrażać narzędzia do obserwacji, priorytetyzować bezpieczeństwo i zgodność oraz korzystać z zarządzanych tożsamości i buforowania.

* W celu zapewnienia obserwowalności wdrażamy narzędzia do monitorowania, rejestrowania i śledzenia, takie jak MLflow, Traceloop i Prompt flow, aby zrozumieć zachowanie modelu, diagnozować problemy i poprawiać doświadczenia użytkownika.

* LLMOps to specjalistyczna dziedzina w ramach MLOps, która koncentruje się na zarządzaniu unikalnymi wyzwaniami i ryzykiem związanym z modelami LLM. Obie dziedziny mają wspólne cele,

takie jak automatyzacja, powtarzalność, monitorowanie i bezpieczeństwo, ale różnią się wymaganiami dotyczącymi danych, złożonością modelu i charakterystyką wyników. LLMOps rozwiązuje unikalne wyzwania, takie jak stronniczość, halucynacje i kwestie etyczne związane z modelami LLM.