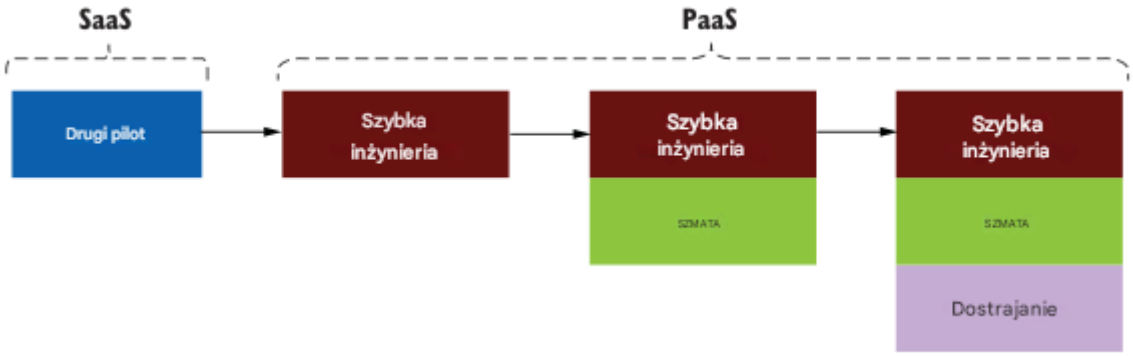
Dopasowanie modeli poprzez adaptację i dostrajanie

Bazując na złożonym świecie dużych modeli językowych (LLM), kluczowym aspektem, który wysuwa się na pierwszy plan w praktycznym wdrażaniu sztucznej inteligencji (AI), jest koncepcja adaptacji modelu. W kontekście LLM adaptacja modelu polega na modyfikacji wstępnie wytrenowanego modelu, takiego jak GPT-3.5 Turbo, w celu zwiększenia jego wydajności w określonych zadaniach lub zbiorach danych. Ten proces jest ważny, ponieważ chociaż wstępnie wytrenowane modele oferują szerokie rozumienie języka i kontekstu, mogą one sprawdzać się w specjalistycznych zadaniach tylko po adaptacji. Adaptacja modelu obejmuje szereg technik, z których każda ma na celu dostosowanie rozległej wiedzy ogólnej modelu do konkretnych zastosowań. Ścieżka adaptacji modelu nie polega jedynie na zwiększeniu wydajności, ale na przekształceniu ogólnego modelu AI w wyspecjalizowane narzędzie, które sprawnie radzi sobie ze zniuansowanymi wymaganiami rozwiązań korporacyjnych. W przypadku przedsiębiorstw adaptacja umożliwia LLM-om radzenie sobie z żargonem branżowym, przestrzeganie norm regulacyjnych w niektórych przypadkach oraz dostosowanie się do unikalnych kontekstów operacyjnych firm. Ta istotność jest kluczowa dla wdrażania rozwiązań AI, które dodają wartości do środowisk korporacyjnych. Należy zauważyć, że większość organizacji powinna powstrzymać się od przechodzenia bezpośrednio do dostrajania. Musimy traktować to jako kontinuum różnych technik, nakładających się na siebie i wzajemnie uzupełniających; co więcej, nie wykluczają się one wzajemnie. Wiele takich technik omówiliśmy już w książce. Dla większości organizacji, jeśli w używanej aplikacji istnieje oferta SaaS, taka jak copilot, jest to najlepszy punkt wyjścia. Ta aplikacja wykorzystuje gotowe rozwiązania SaaS wdrożone w GenAI i zapewnia maksymalny zwrot z inwestycji (ROI). W scenariuszach, w których rozwiązanie SaaS nie jest dostępne ani odpowiednie, a preferowane jest podejście PaaS, zaleca się rozpoczęcie od inżynierii natychmiastowej jako kroku podstawowego i rozwijanie go. Gdy będziemy musieli ugruntowywać generacje modeli na podstawie naszych danych, zastosujemy generację wspomaganą wyszukiwaniem (RAG) w połączeniu z inżynierią natychmiastową, jak pokazano na rysunku.



W przypadku stosowania zaawansowanych modeli granicznych, takich jak GPT-4, takie połączenie rozwiązuje 95% przypadków biznesowych przedsiębiorstw. W pewnym momencie tego kontinuum przedsiębiorstwa osiągną punkt, w którym zajdzie potrzeba dostrojenia modelu do określonych wymagań. Nawet dostrojenie nie eliminuje potrzeby stosowania inżynierii natychmiastowej i RAG. Przyjrzymy się temu przypadkowi w tym rozdziale, gdy dostrajamy i wykorzystujemy model, który nadal wymaga inżynierii natychmiastowej, aby uzyskać pożądane rezultaty. W tym rozdziale opisano różne techniki adaptacji modelu, pomagając nam zrozumieć związane z nimi wyzwania, zobaczyć, jak przedsiębiorstwa mogą wdrażać aplikacje, a na koniec dostroić i wdrożyć model w środowisku produkcyjnym. Zacznijmy od zrozumienia, czym jest adaptacja modelu.

Czym jest adaptacja modelu?

Adaptacja modelu to dostosowanie modelu LLM do lepszego wykonywania określonego zadania w określonej dziedzinie i jest bardzo podobna do transferu wiedzy. Oba podejścia polegają na wykorzystaniu wstępnie wytrenowanego modelu jako punktu wyjścia. Modele te są zazwyczaj trenowane na dużych zbiorach danych i wypracowały solidne zrozumienie różnych cech i wzorców. Kluczową ideą adaptacji modelu i transferu wiedzy jest zastosowanie modelu wytrenowanego na jednym zadaniu do innego, ale powiązanego zadania. Oszczędza to czas i zasoby, które w przeciwnym razie byłyby potrzebne do trenowania modelu od podstaw. Jak wiemy, modele LLM są trenowane na dużej ilości ogólnych danych tekstowych, co zapewnia im szerokie rozumienie języka. Mimo to mogą nie nadawać się do niektórych zadań lub dziedzin wymagających specjalistycznej wiedzy lub słownictwa. Główną ideą adaptacji modelu jest to, że wiedza zdobyta z pierwotnego zadania może pomóc w wykonaniu nowego zadania. Ogólnie rzecz biorąc, istnieją dwie szerokie kategorie adaptacji modelu – dziedzinowa i zadaniowa:

* Adaptacja dziedzinowa — jeśli masz model wytrenowany w jednej domenie (np. ogólne artykuły prasowe) i chcesz, aby dobrze radził sobie w innej, ale powiązanej domenie (np. artykuły medyczne), zastosujesz techniki adaptacji dziedzinowej.

* Adaptacja zadaniowa — jeśli masz model wytrenowany w jednym zadaniu (np. analiza sentymentu) i chcesz, aby wykonywał nowe, ale powiązane zadanie (np. wykrywanie emocji), można zastosować techniki adaptacji zadaniowej.

Na przykład, model LLM wytrenowany na artykułach z Wikipedii może słabo radzić sobie z pytaniami medycznymi lub dokumentami prawnymi. Dlatego adaptacja modelu jest niezbędna do dostrojenia modelu LLM do mniejszego, specyficznego dla zadania lub domeny zbioru danych, co pomaga modelowi nauczyć się odpowiednich wzorców i cech dla docelowego zadania lub domeny.

Podstawy adaptacji modelu

Adaptacja modelu w modelach LLM polega na udoskonaleniu wstępnie wytrenowanego modelu w celu lepszego dopasowania go do konkretnych zadań lub danych. Koncepcję tę można ogólnie podzielić na dwie główne kategorie:

* Pełne dostrajanie — to podejście aktualizuje wszystkie parametry modelu LLM. Obejmuje ono kompleksowe ponowne trenowanie modelu na nowych danych, wprowadzając istotne zmiany w jego wyuczonych wzorcach.

* Adaptacja niskiej rangi — w przeciwieństwie do pełnego dostrajania, adaptacja niskiej rangi koncentruje się na modyfikacji mniejszego zestawu parametrów modelu. Ta metoda wprowadza macierze trenowalne do każdej warstwy modelu LLM, skutecznie zmniejszając liczbę parametrów wymagających dostosowania. Ta sekcja skupi się głównie na tej kategorii adaptacji modelu.

Przyjrzyjmy się kluczowym technikom leżącym u podstaw adaptacji modelu:

* Transfer learning — ta strategia uczenia maszynowego (ML) polega na zastosowaniu modelu wytrenowanego dla jednego zadania do innego, ale powiązanego zadania. Na przykład model wytrenowany na tekście angielskim może zostać zaadaptowany do pracy z tekstem francuskim. Transfer learning polega na wykorzystaniu wiedzy z jednej dziedziny do poprawy wydajności w innej.
* Dostrajanie — Dostrajanie kontynuuje trenowanie wstępnie wytrenowanego modelu na nowym, zazwyczaj mniejszym i bardziej wyspecjalizowanym zbiorze danych. Subtelnie dostosowuje parametry

modelu, aby dopasować jego wiedzę do nowego zadania lub danych. W zależności od zadania, danych i konkretnego modelu LLM, można zastosować różne techniki adaptacji modelu:

* Moduły specyficzne dla zadania — Ta technika dodaje moduł (taki jak klasyfikator lub dekoder) do modelu LLM, dostosowany do konkretnego zadania. Zarówno moduł, jak i model LLM są następnie dostrajane na podstawie danych specyficznych dla zadania. Pozwala to modelowi LLM poznać zawiłości konkretnego zadania, zachowując jednocześnie szeroką wiedzę lingwistyczną.

* Adaptacja niskiego rzędu (LoRA) — LoRA stosuje przybliżenie niskiego rzędu do modelu LLM i dostraja tylko te komponenty. Ta metoda zmniejsza liczbę parametrów wymagających dostosowania, jednocześnie zachowując wydajność i elastyczność modelu.

* Uczenie federacyjne — to podejście dostraja model LLM w wielu rozproszonych zbiorach danych, umożliwiając modelowi uczenie się na podstawie zróżnicowanych danych, przy jednoczesnym zachowaniu prywatności. Na przykład, uczenie federacyjne może zaadaptować BERT do analizy tekstów medycznych, wykorzystując dane z różnych szpitali, co pozwoli na stworzenie specjalistycznej wersji, takiej jak Med-BERT.

Żadna pojedyncza technika nie ma uniwersalnego zastosowania — kluczem jest eksperymentowanie. Zrozumienie tych niuansów jest kluczowe dla efektywnego wykorzystania adaptacji i dostrajania modelu. Metody te ucieleśniają zasady transferu uczenia i zapewniają praktyczne sposoby na poprawę wydajności i zastosowania modeli AI w różnych scenariuszach.

Zalety i wyzwania dla przedsiębiorstw

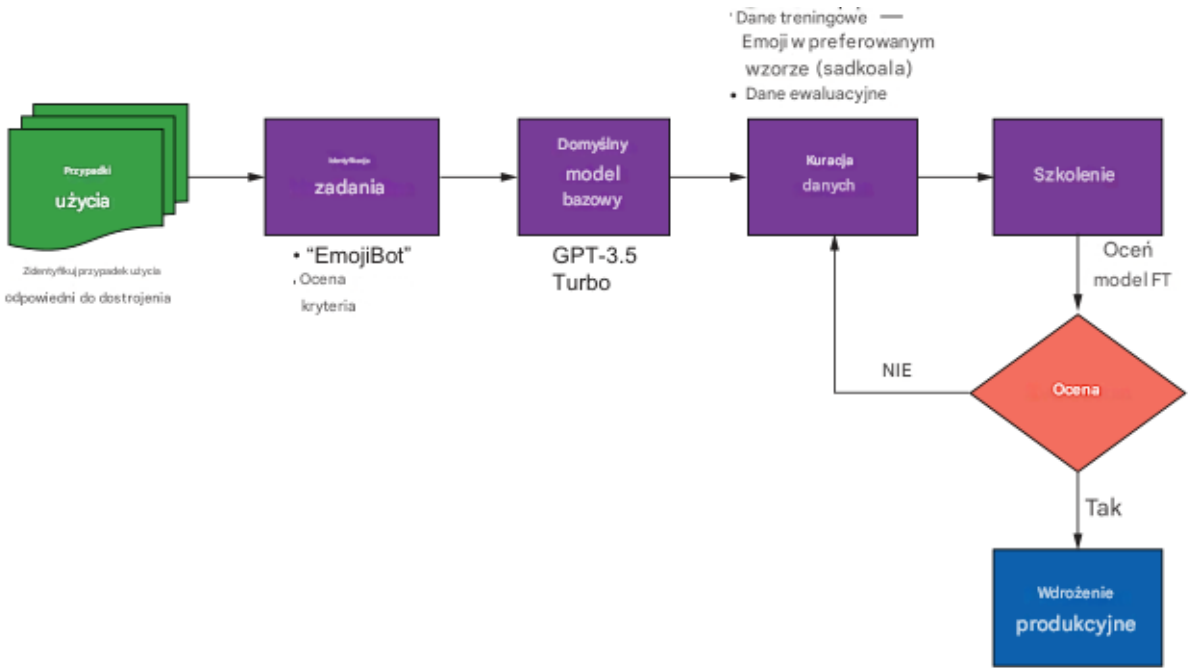
Adaptacja modelu ma coraz większe znaczenie dla przedsiębiorstw w niektórych branżach i scenariuszach. Zapewnia ona znaczne korzyści w zakresie wydajności, konkurencyjności i innowacyjności. Wykorzystując zaadaptowane modele sztucznej inteligencji, firmy mogą osiągać dokładniejsze wyniki w krótszym czasie i przy mniejszych zasobach niż w przypadku tworzenia modeli od podstaw. Na przykład w wysoce wyspecjalizowanych dziedzinach (np. medycynie i farmacji), gdzie margines błędu musi być bliski zeru, dostrojenie modelu do konkretnych zadań jest jednym z niewielu sposobów na osiągnięcie pożądanego rezultatu. Inne wyspecjalizowane obszary, takie jak złożone szczegóły finansowe (np. wykrywanie oszustw) i migracja starszego kodu (np. Cobol itp.), to cenne przykłady, w których przedsiębiorstwa powinny rozważyć dostrojenie modelu. Co więcej, przedsiębiorstwa mogą również lepiej realizować wyspecjalizowane zadania i zyskiwać przewagę konkurencyjną, w zależności od przypadku użycia. Jest to szczególnie istotne w przypadkach, gdy przedsiębiorstwa mają do czynienia z unikalnymi zbiorami danych i wymagają, aby modele rozumiały ich specyficzny kontekst biznesowy. Adaptacja modelu umożliwia personalizację, zwiększając dokładność i trafność analizy nastrojów, przewidywania trendów rynkowych lub spersonalizowanych interakcji z klientami. Korzystając z modeli dostosowanych do swoich specyficznych potrzeb, firmy mogą uzyskać wgląd i zwiększyć wydajność, co zapewni im przewagę konkurencyjną na rynku. Przedsiębiorstwa mogą zwiększyć wydajność i oszczędności kosztów, zmniejszając zapotrzebowanie na zasoby. Dostrajanie istniejących modeli wymaga znacznie mniejszej mocy obliczeniowej i danych w porównaniu z modelami trenowanymi od podstaw, co przekłada się na niższe koszty i krótszy czas wdrożenia. Na przykład, trenowanie modelu Llama 2 o parametrach 70B zajęło wiele miesięcy i 1 720 320 godzin pracy GPU, w porównaniu z dostrajaniem modelu GPT-3.5 Turbo, które zajmuje tylko kilka godzin. Adaptacja modelu wiąże się z wyzwaniami i należy wziąć pod uwagę kilka kluczowych obszarów. Po pierwsze, kluczowe są dane specyficzne dla zadania. Niezbędne jest posiadanie wystarczającej ilości danych do dostrojenia modelu LLM, zapewniając, że dane te są czyste, spójne i reprezentatywne dla konkretnego zadania. W zależności od zadania i charakterystyki modelu LLM, dane te mogą wymagać wstępnego przetwarzania, rozszerzenia lub etykietowania. Określenie, ile

danych jest wystarczające do precyzyjnego dostrojenia, może być procesem złożonym, ponieważ różni się w zależności od kilku czynników; co najmniej jest to kilkaset do tysiąca przykładów, w zależności od modelu. Określenie odpowiednich danych dla modeli precyzyjnego dostrajania, takich jak GPT-3.5 firmy OpenAI, zależy od wielu czynników. Złożoność i specyfika zadania mają duży wpływ na wymagania dotyczące danych, a bardziej złożone zadania wymagają większej ilości danych. Jednak jakość danych jest kluczowa i często przeważa nad ich ilością. Większe modele, takie jak GPT-3.5, mogą korzystać z większej ilości danych ze względu na swoją rozległą pojemność, ale mogą również efektywnie uczyć się z mniejszych, wysokiej jakości zbiorów danych. Organizacje zazwyczaj zaczynają od bazowego zbioru danych i dostosowują go w oparciu o wydajność modelu, który jest stale monitorowany pod kątem oznak nadmiernego lub niedostatecznego dopasowania. Ograniczenia praktyczne, takie jak zasoby obliczeniowe i czas, również odgrywają rolę w określaniu rozmiaru zbioru danych. Doświadczenie i wiedza specjalistyczna analityków danych często wpływają na decyzję. Analiza porównawcza i ciągła ocena są niezbędne do znalezienia optymalnej równowagi między ilością i jakością danych, dostosowanej do konkretnych wymagań zadania. Kolejnym istotnym wyzwaniem są zasoby obliczeniowe i koszty. Dostrajanie modeli LLM może być zasobochłonne i kosztowne, często wymagając znacznej mocy obliczeniowej (w szczególności procesorów GPU) połączonej z szybką pamięcią. Aby sobie z tym poradzić, konieczne może być korzystanie z usług w chmurze, inwestycja w specjalistyczny sprzęt lub wdrożenie systemów rozproszonych. Ponadto koszt dostępu do wstępnie wytrenowanych modeli LLM może się różnić w zależności od dostawcy i umów licencyjnych, co może zwiększyć całkowity koszt. Wydajność i generalizacja są również kluczowymi czynnikami. Ocena wydajności dostrojonego modelu LLM jest niezbędna; polega ona na porównaniu go z innymi modelami lub ustalonymi punktami odniesienia, co zapewnia, że dostrojony model LLM nie będzie nadmiernie dopasowany do danych treningowych i będzie mógł dobrze generalizować na nowe lub nieznane dane wejściowe. Należy również uwzględnić etyczne i społeczne implikacje stosowania dopracowanych programów LLM. Obejmuje to zrozumienie potencjalnych zagrożeń i błędów, takich jak obawy związane z prywatnością danych, rzetelnością modeli i skutkami społecznymi. Przestrzeganie odpowiednich wytycznych, standardów lub przepisów jest niezbędne, aby zapewnić etyczne i odpowiedzialne korzystanie z dopracowanych programów LLM. Wreszcie, kluczowe jest znalezienie odpowiednich talentów. Potrzeba wyspecjalizowanych talentów i wiedzy specjalistycznej jest istotnym czynnikiem w skutecznym dopracowaniu programów LLM, w tym osób dogłębnie rozumiejących uczenie maszynowe (ML), przetwarzanie języka naturalnego (NLP) i specyficzną architekturę programów LLM. Eksperci ci muszą posiadać umiejętności w różnych dziedzinach, takich jak przygotowywanie danych, projektowanie architektury modeli, strategie szkoleniowe i ocena wydajności. Potrzeba wykwalifikowanego personelu dodaje kolejny poziom wyzwań do i tak już złożonego procesu dostrajania programów LLM.

Kiedy dostrajać program LLM

Dostrajanie to technika mająca na celu poprawę wydajności modelu w określonym zadaniu. Powinna być jednak ostateczną opcją i stosowana dopiero po zastosowaniu innych technik, takich jak inżynieria błyskawiczna (Szybkie dostrajanie) i RAG. Techniki te wzajemnie się uzupełniają i powinny być łączone w celu uzyskania najlepszych wyników, nawet przy użyciu precyzyjnie dostrojonych modeli. Jak widzieliśmy w poprzednich rozdziałach, inżynieria natychmiastowa i RAG nie wykluczają się wzajemnie, ale uzupełniają się i powinny być łączone w celu uzyskania najlepszych wyników, nawet przy użyciu precyzyjnie dostrojonych modeli. To podejście oparte na łączeniu w stosy daje najlepsze wyniki, nawet przy użyciu precyzyjnie dostrojonych modeli. Po podjęciu decyzji o dostrojeniu modelu przygotowujemy zbiór danych potrzebny do treningu i rozpoczynamy proces dostrajania, który może trwać od kilku godzin do kilku dni. Po treningu porównujemy precyzyjnie dostrojony model z modelem bazowym i bazą danego zadania. Posłużmy się przykładem, aby pomóc nam w dostrojeniu i

zrozumieniu różnych aspektów. Załóżmy, że chcemy dostosować model do reagowania za pomocą emoji — bota, który rozumie, o co pytamy, ale odpowiada wyłącznie za pomocą emoji. Nazwiemy go EmojiBot. Chcemy dostroić GPT-3.5 Turbo i przekształcić go w EmojiBota. Aby jednak pokazać, że te emoji są różne i wyspecjalizowane dla danego zadania, nie chcemy emoji, których spodziewalibyśmy się zobaczyć, powiedzmy, w aplikacji czatu, w mediach społecznościowych lub w naszych tekstach. Chcemy raczej tych, które są zgodne z formatem używanym przez Microsoft Teams. Rysunek 2 przedstawia ogólny przebieg dostrajania.



Najpierw identyfikujemy zadanie, które skorzystałoby na dostrajaniu (takie jak EmojiBot). Identyfikujemy, które cechy nie spełniają wymagań zadania i tworzymy kryteria oceny. Następnie porównujemy wydajność domyślnych modeli z naszymi potrzebami. Jeśli działają dobrze, ustalamy punkt odniesienia i tworzymy zbiór danych wymagany do dostrajania. Ilość i format danych zależą od modelu; szczegóły omówimy później. Uzyskujemy dostrojony model po treningu, który może trwać godziny lub dni, w zależności od zadania. Następnie musimy go ocenić w porównaniu z modelem bazowym i punktem odniesienia dla konkretnego zadania, używając miar jakościowych i ilościowych. Dość często, a wręcz można się tego spodziewać, pierwszy dostrojony model będzie gorszy od modelu domyślnego. Zazwyczaj znalezienie odpowiedniego modelu wdrożenia wymaga 10–12 iteracji treningowych. Każda iteracja wymaga modyfikacji danych treningowych w celu wyeliminowania słabych punktów, co może trwać od kilku godzin do kilku dni. Jest to proces czasochłonny i wymagający wysiłku, który powinien być jednym z ostatnich kroków.

UWAGA: Dostrajanie poprawia wydajność modelu w zadaniach podobnych do tych opisanych w zbiorze danych dostrajających. Proces ten może objawiać się zwiększoną dokładnością, bardziej trafnymi odpowiedziami lub lepszym zrozumieniem języka specyficznego dla danej dziedziny. Poprawa wydajności w przypadku tańszych lub szybszych modeli jest dodatkową zaletą, a nie czymś gwarantowanym. Jednym ze sposobów osiągnięcia tego celu jest dostrojenie mniejszego modelu, takiego jak GPT-3.5 Turbo, do konkretnego zadania w celu jego ulepszenia, zamiast stosowania droższego i bardziej wydajnego modelu, takiego jak GPT-4. Teraz, gdy zidentyfikowaliśmy zadanie, które ma sens w kontekście dostrajania, czyli EmojiBot, który ma odpowiadać za pomocą emotikonów, ale według określonego wzorca, przyjrzyjmy się krokom niezbędnym do dostrojenia LLM, takiego jak GPT-3.5 Turbo.

Kluczowe etapy dostrajania LLM

Chcąc dostroić model do zidentyfikowanego zadania, należy wyróżnić pięć kluczowych etapów:

1. Wybór modelu i metody dostrajania — Aby dostroić model językowy, należy wybrać model podstawowy, który odpowiada zadaniu i danym. Dostępne są różne modele, takie jak GPT, BERT i RoBERTa. Należy wziąć pod uwagę takie czynniki, jak przydatność modelu do zadania, rozmiar danych wejściowych/wyjściowych, rozmiar zbioru danych oraz infrastrukturę techniczną. Metody dostrajania mogą się różnić w zależności od zadania i danych, na przykład transfer learning, dostrajanie sekwencyjne lub dostrajanie specyficzne dla danego zadania.

2. Kuracja danych — Ten etap obejmuje przygotowanie zbioru danych specyficznego dla danego zadania do dostrajania i w dużej mierze polega na przygotowaniu i wstępnym przetworzeniu zbioru danych. Proces ten często obejmuje czyszczenie danych, normalizację tekstu (np. tokenizację) oraz konwersję danych do formatu zgodnego z wymaganiami wejściowymi modelu LLM (np. etykietowanie danych). Konieczne jest upewnienie się, że dane reprezentują zadanie i domenę oraz obejmują szereg scenariuszy, z którymi model ma się spotkać w środowisku produkcyjnym.

3. Dostrajanie — Ten etap to faktyczny proces dostrajania, który obejmuje trenowanie wstępnie wytrenowanego modelu LLM na zbiorze danych specyficznym dla danego zadania. Proces trenowania obejmuje optymalizację wag i parametrów modelu w celu zminimalizowania funkcji straty i poprawy jego wydajności w danym zadaniu. Proces dostrajania może obejmować kilka rund trenowania na zbiorze treningowym, walidację zbioru walidacyjnego oraz dostrajanie hiperparametrów w celu optymalizacji wydajności modelu.

4. Ewaluacja — Po zakończeniu procesu dostrajania należy ocenić wydajność modelu na zbiorze danych testowych. Pomaga to upewnić się, że model dobrze generalizuje na nowe dane i dobrze działa w konkretnym zadaniu. Do typowych wskaźników używanych do oceny należą dokładność, precyzja, przypominanie, wynik F1, ocena dwujęzyczna (BLEU), ocena zorientowana na przypominanie (ROUGE) itd.

5. Wdrożenie (wnioskowanie) — po ocenie dostrojonego modelu i uzyskaniu satysfakcjonującej wydajności można go wdrożyć w środowisku produkcyjnym. Proces wdrażania może obejmować integrację modelu z większym systemem, skonfigurowanie niezbędnej infrastruktury oraz monitorowanie wydajności modelu w rzeczywistych scenariuszach.

Skoro znamy już podstawową koncepcję adaptacji modelu i wiemy, kiedy należy go dostrajać, zobaczmy, jak to zrobić.

Dostrajanie modeli OpenAI

W tym przykładzie dostroimy model GPT-3.5 Turbo w OpenAI. Obecnie w OpenAI dostępne są tylko GPT-4, GPT-3.5 Turbo, GPT-3 Babbage (Babbage-002) i GPT-3 (Davinci-002) do dostrajania. Kilka modeli LLM OSS, takich jak Llama 2 firmy Meta i Falcon firmy G42, można dostroić. Aby było to jak najbardziej praktyczne dla organizacji, pokażemy ten proces z wykorzystaniem zarówno Azure OpenAI, jak i OpenAI. Chcemy dopracować GPT-3.5 Turbo i przekształcić go w EmojiBota, w którym model będzie reagował wyłącznie za pomocą emotikonów. Jednak, jak wspomnieliśmy wcześniej, chcemy, aby emotikony odpowiadały formatowi używanemu w Microsoft Teams. W Microsoft Teams tekst w nawiasach, taki jak (pies), renderuje odpowiednie emotikony. Dopracujemy model tak, aby reagował na ten tekst, który reprezentuje konkretne zadanie, które chcemy ulepszyć. Mając już zadanie, zacznijmy przygotowywać zbiór danych.

Przygotowanie zbioru danych do dostrajania

Po osiągnięciu punktu, w którym zidentyfikowaliśmy zadanie, dla którego dostrajanie ma sens, musimy utworzyć zbiór danych przykładów wymaganych do dostrajania. Musimy utworzyć dwa zbiory danych: jeden do trenowania, a drugi do walidacji. Zbiór danych walidacyjnych to podzbiór danych używany do oceny wydajności dostrojonego modelu w zadaniu docelowym. Różni się on od zbioru danych treningowych, który służy do aktualizacji parametrów modelu, oraz zbioru danych testowych, który służy do pomiaru końcowej dokładności modelu. Zbiór danych walidacyjnych jest ważny dla dostrajania modeli LLM, ponieważ pomaga uniknąć przeuczenia, które występuje, gdy model uczy się określonych wzorców danych treningowych i nie jest w stanie generalizować na nowe dane. Korzystając ze zbioru danych walidacyjnych, można monitorować postęp modelu i dostosowywać tempo uczenia, liczbę epok lub inne hiperparametry w celu optymalizacji wydajności modelu. Te przykłady powinny pokazać różne sposoby rozwiązania problemu i wyniki każdej metody. Musimy również zidentyfikować niedociągnięcia przy użyciu modelu bazowego, takie jak niespójna wydajność w przypadkach brzegowych, brak możliwości umieszczenia wystarczającej liczby komunikatów o strzałach w oknie kontekstowym, aby sterować modelem, wysokie opóźnienie itd. Zdecydowanie zaleca się użycie zbioru danych walidacyjnych do pomiaru skuteczności dostrajania. Zbiory danych treningowych i walidacyjnych są w formacie JSONL, przy czym każdy wiersz zawiera obiekt JSON z kluczem tekstowym dla tekstu wejściowego i kluczem docelowym dla pożądanego tekstu wyjściowego. Dostrojone modele są bezpośrednio skorelowane z wysokiej jakości danymi treningowymi. Różne modele wymagają różnej ilości danych treningowych. Do efektywnego treningu potrzebujemy setek, a nawet tysięcy starannie wyselekcjonowanych przykładów danych. Chociaż API wymaga co najmniej 10 przykładów, zazwyczaj lepiej jest mieć więcej. Dziesięć przykładów nie wystarczy, aby znacząco wpłynąć na modele LLM, takie jak GPT-3.5 Turbo. OpenAI zaleca posiadanie co najmniej 50 dobrych przykładów do trenowania naszego modelu. Zalecają również więcej dobrych przykładów dla modeli o lepszym dostrojeniu niż złych, ponieważ mogą one negatywnie wpłynąć na model. W związku z tym zaleca się korzystanie tylko z najlepszych przykładów z danych wewnętrznych. Poniższa lista przedstawia przykładowy plik JSONL dla danych czatu.

Listing 1: Przykład JSONL

```
{"messages": [{"role": "system", "content": "You're a chatbot that only
responds with emojis!"}, {"role": "user", "content": "I heard the koala
habitat was destroyed by the fire."}, {"role": "assistant", "content":
"(sadkoala)" }]}
{"messages": [{"role": "system", "content": "You're a chatbot that only
responds with emojis!"}, {"role": "user", "content": "I've been working
nonstop and need a break."}, {"role": "assistant", "content": "(tired)"
}]}
{"messages": [{"role": "system", "content": "You're a chatbot that only
responds with emojis!"}, {"role": "user", "content": "I just finished
reading an amazing book!"}, {"role": "assistant", "content": "(like)"
}]}
```

Jak widać, modelowi pokazuje się, jak reagować za pomocą emoji sformatowanych według określonego wzorca, takich jak (smutna koala), (zmęczony) i (lubię to).

PODSTAWOWE SPRAWDZENIA

Przed dostrojeniem ważne jest przeprowadzenie podstawowych sprawdzeń danych treningowych, aby uniknąć marnowania czasu i zasobów. Kontrole te mogą obejmować czytelność danych, walidację formatowania, prostą analizę brakujących par i długość tokena. Plik danych jest weryfikowany poprzez załadowanie i odczytanie za pomocą funkcji basic_checks(). Przyjmuje ona nazwę pliku jako dane wejściowe i zwraca liczbę znalezionych wiadomości. Wiadomości muszą być w formacie uzupełniania czatu, aby dostroić GPT-3.5 Turbo.

Listing 2: Walidacja zbioru danych: podstawowe kontrole

```
# Basic checks to ensure the data file is valid
def basic_checks(data_file):
    try:
        with open(data_file, 'r', encoding='utf-8') as f:
            dataset = [json.loads(line) for line in f]

        print(f"Basic checks for file {data_file}:")
        print("Count of examples in training dataset:", len(dataset))
        print("First example:")
        for message in dataset[0]["messages"]:
            print(message)
        return True
    except Exception as e:
        print(f"An error occurred in file {data_file}: {e}")
        return False
```

Opens the file in read-mode

Loads each line of the file as a JSON object and stores it in a list

Prints the first example from the dataset and helps visually check whether things intuitively look OK

Loops through the messages in the first example and prints each one

SPRAWDZANIE FORMATU

Po wykonaniu podstawowych sprawdzeń, następnym krokiem jest sprawdzenie pliku pod kątem formatu i upewnienie się, że jest on poprawnie ustrukturyzowany przed dalszym przetwarzaniem. Jest to ważny krok, głównie dlatego, że nawet jeśli format jest niepoprawny, nie pojawi się błąd na początku zadania szkoleniowego, ale wynikowy model będzie bardzo słaby i zauważymy to dopiero po szkoleniu, podczas wdrożenia. Aby uniknąć wielu z tych problemów, zdecydowanie zaleca się sprawdzenie formatów. Listing 3 przedstawia funkcję format_checks(), która sprawdza format i parowanie ukończonych rozmów, z zestawem danych i nazwą pliku jako dwoma argumentami. Wychwytuje większość błędów, ale nie wszystkie. Funkcja iteruje po każdym przykładzie w zbiorze danych i sprawdza typy danych, obecność list wiadomości i kluczy wiadomości. Sprawdza, czy posiada on odpowiednie role i walidację treści. Ta funkcja pomaga również w debugowaniu problemów związanych z danymi.

Listing 3: Walidacja zbioru danych: sprawdzanie formatu

```
def format_checks(dataset, filename):

# Initialize a dictionary used to track format errors

format_errors = defaultdict(int)

# Iterate over each example in the dataset

for ex in dataset:

# Check if the example is a dictionary, if not
```

```
# increment the corresponding error count
if not isinstance(ex, dict):
format_errors["data_type"] += 1
continue
# Check if the example has a "messages" key,
# if not increment the corresponding error count
messages = ex.get("messages", None)
if not messages:
format_errors["missing_messages_list"] += 1
continue
# Iterate over each message
for message in messages:
# Check if the message has "role" and "content" keys,
# if not increment the corresponding error count
if "role" not in message or "content" not in message:
format_errors["message_missing_key"] += 1
# Check if the message has any unrecognized keys,
# if so increment the corresponding error count
if any(k not in ("role", "content", "name",
➥"function_call") for k in message):
format_errors["message_unrecognized_key"] += 1
# Check if the role of the message is one of the recognized
# roles, if not increment the corresponding error count
if message.get("role", None) not in (
"system",
"user",
"assistant",
"function",
):
format_errors["unrecognized_role"] += 1
# Check if the message has either content or a function call,
```

```
# and if the content is a string, if not increment the
# corresponding error count
content = message.get("content", None)
function_call = message.get("function_call", None)
if (not content and not function_call) or not
➥isinstance(content, str):
format_errors["missing_content"] += 1
# Check if there is at least one message with the role "assistant",
# if not increment the corresponding error count
if not any(message.get("role", None) == "assistant"
➥for message in messages):
format_errors["example_missing_assistant_message"] += 1
# If there are any format errors, print them and return False
if format_errors:
print(f"Formatting errors found in file {filename}:")
for k, v in format_errors.items():
print(f"{k}: {v}")
return False
print(f"No formatting errors found in file {filename}")
return True
```

Na koniec powinniśmy również zrozumieć, jak zbiór danych zachowuje się w kontekście prostych rozkładów danych, liczby tokenów i kosztów.

UWAGA

Liczba tokenów jest ważna nie tylko ze względu na koszt. Jeśli jest większa niż maksymalna liczba tokenów, jaką model może obsłużyć, zostanie ona obcięta bez ostrzeżenia. Wiedza o tym z góry jest bardzo pomocna. Poniższa lista pokazuje, jak możemy zakończyć sprawdzanie zbioru danych.

Listing 4: Walidacja zbioru danych: Oszacowanie kosztów i analiza podstawowa

```
# Pricing and default n_epochs estimate
MAX_TOKENS = 4096
TARGET_EPOCHS = 3
MIN_TARGET_EXAMPLES = 100
MAX_TARGET_EXAMPLES = 25000
```

```
MIN_DEFAULT_EPOCHS = 1
MAX_DEFAULT_EPOCHS = 25
def estimate_tokens(dataset, assistant_tokens):
# Set the initial number of epochs to the target epochs
n_epochs = TARGET_EPOCHS
# Get the number of examples in the dataset
n_train_examples = len(dataset)
# If the examples total is less than the minimum target
# adjust the epochs to ensure we have enough examples for
# training
if n_train_examples * TARGET_EPOCHS < MIN_TARGET_EXAMPLES:
n_epochs = min(MAX_DEFAULT_EPOCHS, MIN_TARGET_EXAMPLES
➥// n_train_examples)
# If the number of examples is more than the maximum target
# adjust the epochs to ensure we don't exceed the maximum
# for training
elif n_train_examples * TARGET_EPOCHS > MAX_TARGET_EXAMPLES:
n_epochs = max(MIN_DEFAULT_EPOCHS, MAX_TARGET_EXAMPLES
➥// n_train_examples)
# Calculate the total number of tokens in the dataset
n_billing_tokens_in_dataset = sum(
min(MAX_TOKENS, length) for length in assistant_tokens
)
# Print the total token count that will be charged during training
print(
f"Dataset has ~{n_billing_tokens_in_dataset} tokens that
➥will be charged for during training"
)
# Print the default number of epochs for training
print(f"You will train for {n_epochs} epochs on this dataset")
# Print the total number of tokens that will be charged during training
```

```
print(f"You will be charged for ~{n_epochs *
➥n_billing_tokens_in_dataset} tokens")
# If the total token count exceeds the maximum tokens, print a warning
if n_billing_tokens_in_dataset > MAX_TOKENS:
print(
f"WARNING: Your dataset contains examples longer than
➥4K tokens by {n_billing_tokens_in_dataset –
➥MAX_TOKENS} tokens."
)
print(
"You will be charged for the full length of these
➥examples during training, but only the first
➥4K tokens will be used for training."
```

Ocena LLM

Ocena LLM jest ważna dla zapewnienia ich jakości, rzetelności i uczciwości. Jednak ocena LLM jest złożona, ponieważ obejmuje wiele wymiarów i wyzwań. Utrzymywanie zróżnicowanych automatycznych metryk może pomóc w efektywnym śledzeniu ulepszeń modelu podczas cykli adaptacji, jednocześnie ograniczając kosztowne, ręczne przeglądy. Metryki powinny być dostosowane do przypadków użycia każdego zaadaptowanego modelu i potrzeb biznesowych. Ciągłe rejestrowanie danych z systemów produkcyjnych umożliwia ocenę rzeczywistej wydajności w czasie. Benchmarking z poziomami bazowymi jest niezbędnym krokiem w ocenie dostrojonych modeli GPT. Polega on na porównaniu wydajności dostrojonego modelu z predefiniowanym modelem standardowym lub bazowym. Poziomem bazowym może być wydajność modelu przed dostrojeniem lub inny model znany ze swojej skuteczności w podobnym zadaniu. Celem tego porównania jest ilościowe określenie usprawnień wprowadzonych dzięki dostrojeniu. Na przykład, dostrojony model może zostać porównany ze standardowym modelem tłumaczenia w zadaniu tłumaczenia językowego w celu oceny dokładności tłumaczenia lub poprawy płynności. Proces ten pomaga zrozumieć skuteczność dostrajania i zidentyfikować obszary, w których model został ulepszony lub nadal wymaga udoskonalenia.

KRYTERIA OCENY

Przygotowując zbiór danych do dostrajania, należy również zdefiniować kryteria oceny. Podczas dostrajania proces oceny rozpoczyna się od ustalenia jasnych kryteriów kluczowych dla oceny wydajności i skuteczności modelu w jego docelowym zastosowaniu. Kryteria te często obejmują trafność, spójność, dokładność i płynność językową .

Kryteria oceny: Opis

Trafność: Mierzy, jak dobrze odpowiedzi lub wyniki modelu są zgodne z kontekstem i intencją danych wejściowych. Jest to szczególnie istotne w aplikacjach takich jak chatboty, gdzie udzielanie

kontekstowo odpowiednich odpowiedzi jest kluczem do satysfakcji użytkownika. Trafność jest często oceniana poprzez sprawdzenie, czy model jest w stanie pozostać w temacie i dostarczać informacje lub odpowiedzi bezpośrednio odnoszące się do zapytań lub zadań.

Spójność: Odnosi się do logicznej spójności wyników modelu. Dopracowany model powinien generować tekst istotny kontekstowo, logicznie poprawny i spójny. Oznacza to, że odpowiedzi powinny być zgodne z logiczną strukturą i narracją, sensownie wpisując się w kontekst rozmowy lub tekstu. Spójność jest niezbędna do utrzymania zaangażowania użytkownika i zapewnienia, że wyniki modelu są zrozumiałe i znaczące.

Dokładność: Ma to szczególne znaczenie, gdy model jest wykorzystywany do zadań wymagających informacji faktograficznych, takich jak narzędzia edukacyjne, boty informacyjne lub dowolne aplikacje, w których dostarczanie poprawnych informacji ma kluczowe znaczenie. Dokładność mierzy się na podstawie tego, jak dobrze odpowiedzi modelu są zgodne z poprawnością faktyczną i obiektywną prawdą.

Płynność językowa: Odnosi się do gramatycznej i składniowej poprawności wyników modelu. Nawet jeśli model jest wysoce trafny, spójny i precyzyjny, słaba płynność językowa może znacząco pogorszyć komfort użytkowania. Obejmuje to poprawną gramatykę, interpunkcję i styl, zapewniając, że generowany tekst jest poprawny i czytelny dla użytkownika końcowego.

Ocena precyzyjnie dostrojonego modelu GPT z wykorzystaniem tych kryteriów obejmuje połączenie zautomatyzowanych metryk, ręcznej weryfikacji i opinii użytkowników, co zapewnia, że model spełnia wysokie standardy wymagane dla jego konkretnego zastosowania.

WYBÓR ODPOWIEDNICH METRYK

Podczas dostrajania modeli, wybór odpowiednich metryk do oceny ma kluczowe znaczenie dla dokładnej oceny wydajności modelu i ulepszeń [1]. Po dostrojeniu metryki te wskazują, jak dobrze model dostosowuje się do określonych zadań lub domen. Dostarczają one wglądu w różne aspekty wydajności modelu, takie jak dokładność prognoz, jakość języka i efektywność w odniesieniu do konkretnego zadania. Przedsiębiorstwa powinny w miarę możliwości poszukiwać zautomatyzowanej oceny metryk i dysponować zestawem metryk ilościowych i jakościowych.

Metryki ilościowe:

– Kilka metryk pomaga zmierzyć stopień nakładania się wyników modelu na ludzkie teksty referencyjne. W następnej sekcji omówiono niektóre z nich (BLEU, ROUGE, METEOR itp.).

– Wynik F1 ocenia kompromis między dokładnością a wykrywalnością.

– Perplexity ocenia niepewność/pewność modelu dla wygenerowanego tekstu.

– Ukończenie zadania jest wykorzystywane w systemach dialogowych zorientowanych na cel i określa odsetek pomyślnego rozwiązania zadania.

Metryki jakościowe:

– Płynność – ocena gramatyki i czytelności wyników

– Spójność – logiczna spójność i płynność narracji

– Zwięzłość – unikanie powtarzalnego i nadmiernego tekstu

– Dokładność merytoryczna – unikanie obiektywnych fałszerstw

Wybór metryk powinien być zgodny z zamierzonym zastosowaniem modelu, niezależnie od tego, czy chodzi o tłumaczenie, podsumowanie, klasyfikację, czy generowanie kreatywnych treści. Metryki takie jak perplexity, wynik BLEU, ROUGE, wynik F1 i ocena człowieka oferują unikalną perspektywę na możliwości modelu, pomagając zapewnić kompleksową i zrównoważoną ocenę wydajności precyzyjnie dostrojonego modelu. Przyjrzyjmy się każdemu z nich bardziej szczegółowo:

* Perplexity – ta metryka jest standardem w modelowaniu języka, używanym do ilościowego określenia, jak dobrze model przewiduje próbkę. Mierzy niepewność modelu językowego w przewidywaniu kolejnego tokena w sekwencji . Niższy wynik perplexity wskazuje, że model jest bardziej pewny i dokładny w swoich przewidywaniach. Jest to szczególnie ważne w procesie dostrajania, ponieważ może odzwierciedlać, jak dobrze model zaadaptował się do nowych stylów lub dziedzin tekstu. Jest to kluczowy wskaźnik oceny usprawnień w zadaniach generowania języka.

* Wynik BLEU (Bilingual Evaluation Understudy) — wynik BLEU ocenia jakość tłumaczenia maszynowego poprzez porównanie go z tłumaczeniami referencyjnymi. Zlicza pasujące grupy słów i oblicza wynik na podstawie tych dopasowań. Wyższy wynik BLEU wskazuje na lepszą jakość tłumaczenia, ale ma ograniczenia i może nie odzwierciedlać dokładności semantycznej ani płynności .

* ROUGE (Recall-Oriented Understudy for Gisting Evaluation) — ROUGE to wskaźnik do automatycznej oceny podsumowań. Mierzy on nakładanie się wyników generowanych komputerowo na podsumowania referencyjne, aby ocenić jakość podsumowania. Różne warianty ROUGE pozwalają na wgląd w aspekty jakości podsumowania.

* Wynik F1 — Wynik F1 jest przydatny w zadaniach klasyfikacyjnych, takich jak analiza sentymentu i kategoryzacja tematów. Wyrównuje on kompromis między precyzją a przypomnieniem, zapewniając pojedynczą miarę dokładności modelu w kategoryzowaniu lub klasyfikowaniu tekstu.

* Ocena człowieka — Pomimo użyteczności zautomatyzowanych metryk, ludzka ocena pozostaje kluczowa, szczególnie w przypadku zadań wymagających subiektywnej oceny, takich jak generowanie historii, pisanie kreatywne i agenci konwersacyjni. Ewaluatorzy mogą dostarczyć wglądu w takie aspekty, jak naturalność tekstu, jego adekwatność, kreatywność, a nawet subtelności humoru czy sarkazmu. Ta jakościowa ocena uzupełnia metryki ilościowe, oferując bardziej holistyczny obraz wydajności modelu.

Ewaluacja specyficzna dla zadania jest niezbędna do pomiaru wydajności modelu w jego zamierzonym zastosowaniu. Polega ona na użyciu różnych metryk i czynników w zależności od zadania. Na przykład modele podsumowujące są oceniane za pomocą wyników ROUGE oraz ludzkich ocen spójności i informatywności podsumowań. Podobnie, modele pytań i odpowiedzi są oceniane pod kątem dokładności i trafności w odniesieniu do zadanych pytań. Ocena ta zapewnia, że model dobrze radzi sobie w ogólnych wskaźnikach oraz jest skuteczny i niezawodny w konkretnym przypadku użycia.

ANALIZA BŁĘDÓW

Analiza błędów jest kluczowym elementem procesu oceny, obejmującym szczegółową analizę miejsc i przyczyn niezadowalających wyników dostrojonego modelu. Analiza ta pomaga zidentyfikować wzorce błędów modelu, które można kategorycznie podzielić na błędy semantyczne, nieścisłości faktyczne lub niespójności językowe. Na przykład, jeśli model konsekwentnie popełnia błędy w rozumieniu określonych typów zapytań lub generuje odpowiedzi z błędami faktycznymi, zostanie to uwypuklone w analizie błędów. Zrozumienie tych wzorców błędów jest kluczowe dla dalszego udoskonalania modelu i wprowadzania ukierunkowanych ulepszeń. Pomaga również zrozumieć ograniczenia modelu

i obszary, w których może wymagać dodatkowych danych lub bardziej zaawansowanych metod dostrajania. Przejdźmy teraz do dostrajania.

Dostrajanie

Teraz, gdy nasz zbiór danych jest gotowy i zweryfikowany, możemy rozpocząć proces dostrajania. W przypadku konieczności dostrajania należy wykonać dwa kroki. Najpierw przesyłamy zbiór danych, nad którym pracowaliśmy w poprzednich sekcjach. Po przesłaniu każdy plik otrzymuje unikalny identyfikator, który musimy zapisać. Ten identyfikator pliku przekazujemy jako jeden z parametrów do zadania dostrajania, aby wiedziało, którego pliku użyć w danym zadaniu. Możemy to zrobić za pomocą interfejsu API lub graficznego interfejsu użytkownika. Zobaczymy, jak to osiągnąć za pomocą zestawu SDK języka Python i Azure AI Studio. Nie pokażę wszystkich kroków w książce o graficznym interfejsie użytkownika, ale szczegóły są dostępne w towarzyszącym repozytorium GitHub pod adresem https://bit.ly/GenAIBook. Zacznijmy od użycia zestawu SDK.

DOSTRAJANIE ZA POMOCĄ SDK

Poniższy listing pokazuje, jak korzystać z SDK i metody files.create(), przekazać nazwę pliku i określić jego przeznaczenie (dostrajanie).

Listing 5: Przesyłanie zestawu danych w celu dostrojenia

```
import os
from openai import AzureOpenAI

API_VERSION = '2023-09-15-preview'      <- This version (or later) is required for fine-tuning.

client = AzureOpenAI(
    api_key=os.getenv('AOAI_FT_KEY'),
    api_version=API_VERSION,
    azure_endpoint = os.getenv('AOAI_FT_ENDPOINT'))   <- Environment variables with the connection details

TRAINING_FILENAME = 'data/emoji_FT_train.jsonl'   <- Dataset that we need to use for training

# Upload the training dataset files
file = client.files.create(
    file=open(TRAINING_FILENAME, "rb"),
    purpose="fine-tune"
)

print("Training file ID:", file.id)
print("Training file name:", file.filename)
```

Po uruchomieniu tego fragmentu kodu otrzymamy następujący wynik z identyfikatorem pliku, który musimy znać, wykonując drugi krok:

Response:

Training file ID: file-ca4c57d7ad814211a2db49e0382c5a77

Training file name: emoji_FT_train.jsonl

Po przesłaniu pliku musimy rozpocząć zadanie dostrajania. W przypadku korzystania z zestawu SDK odbywa się to za pomocą metody fine_tunings.jobs.create(). Funkcja ta wymaga identyfikatora pliku zestawu danych treningowych z poprzednich kroków oraz modelu, który ma zostać użyty. W naszym przypadku chcemy dostroić GPT-3.5 Turbo, a konkretnie model 0613. Określamy również liczbę epok potrzebnych do dostrajania. Na koniec, parametr sufiksu może nam pomóc w śledzeniu i zarządzaniu dostrojonym modelem.

Listing 6: Rozpoczęcie pracy nad dostrajaniem

```
import os
from openai import AzureOpenAI

API_VERSION = '2023-09-15-preview'

# Connect to the servvice
client = AzureOpenAI(
    api_key=os.getenv('AOAI_FT_KEY'),
    api_version=API_VERSION,
    azure_endpoint = os.getenv('AOAI_FT_ENDPOINT'))
```

This version (or later) is required for fine-tuning.

Environment variables with connection details

```
# Begin by creating the fine-tuning job
ft = client.fine_tuning.jobs.create(
    training_file="file-ca4c57d7ad814211a2db49e0382c5a77",
    model="gpt-35-turbo-0613",
    hyperparameters={
        "n_epochs":3
    },
    suffix="emoji"
)
print("Finetuning job ID:", ft.id)
```

The file ID you see will differ from this one.

Ten fragment kodu przesyła zadanie dostrajania, które zostaje umieszczone w kolejce; w zależności od dostępnej pojemności w danym regionie i centrum danych, zadanie zostanie wykonane. Przypominamy, że dzięki Azure OpenAI można mieć wiele regionów, w których dostępne jest dostrajanie. Nasz przykład pokazuje identyfikator zadania z wywołania interfejsu API: Identyfikator zadania dostrajania: ftjob-367ee1995af740a0bf24876221585f7a. W zależności od rozmiaru zestawu danych, modelu, który chcemy dostroić, oraz hiperparametrów dostrajania, zadanie dostrajania może zająć kilka godzin. Interfejs API zadań dostrajania zawiera wywołanie funkcji list(), za pomocą którego możemy wyświetlić wszystkie przesłane zadania dostrajania.

Listing 7: Wylistowanie wszystkich zadań dostrajania

```
import os

from openai import AzureOpenAI

API_VERSION = '2023-09-15-preview'

client = AzureOpenAI(

api_key=os.getenv('AOAI_FT_KEY'),

api_version=API_VERSION,

azure_endpoint = os.getenv('AOAI_FT_ENDPOINT'))

# List all the FT jobs

ft_jobs = client.fine_tuning.jobs.list()

for ft_job in ft_jobs:

print(ft_job.id, ft_job.status)
```

Przykład takiego wyniku przedstawiono na listingu 9.8. Widzimy, że ukończyliśmy dwa zadania dostrajające, o czym świadczy status „zakończone sukcesem”; jedno zadanie ma obecnie status „uruchomione”, co oznacza, że trwa jedno aktywne zadanie dostrajające. Ostatnie zadanie

dostrajające (ftjob-367ee1995...), które właśnie zgłosiliśmy jako oczekujące, oznacza, że zadanie jest w kolejce do uruchomienia w przyszłości.

Listing 8: Wyjście listy zadań dostrajania

ftjob-367ee1995af740a0bf24876221585f7a pending

ftjob-c41a9dc551834a1aa0be8befe788a22b running

ftjob-1a7faac8856d46e48a038c02555fe6e5 succeeded

ftjob-505d5a8bd321406dbf4605b636b0c0cd succeeded

W przypadku konkretnego zadania dostrajania możemy również zobaczyć różne zdarzenia z nim związane. Poniższa lista pokazuje przykład, ponownie wykorzystując identyfikator naszego nowo przesłanego zadania (ftjob-367ee1995...).

Listing 9: Wypisywanie zdarzeń z zadania dostrajania

```
import os

from openai import AzureOpenAI

API_VERSION = '2023-09-15-preview'

client = AzureOpenAI(

api_key=os.getenv('AOAI_FT_KEY'),

api_version=API_VERSION,

azure_endpoint = os.getenv('AOAI_FT_ENDPOINT'))

#List all the FT events for the job from

#earier: ftjob-367ee1995af740a0bf24876221585f7a

ft_job_events = client.fine_tuning.jobs.list_events(

fine_tuning_job_id="ftjob-367ee1995af740a0bf24876221585f7a ",

limit=2)

# Loop through the events and print the details

for ft_job_event in ft_job_events:

print(ft_job_event.id, ft_job_event.message)
```

The output in this case is

ftevent-1e89dc7cc62046048bcea50de1cccbb9 Jobs ahead in queue: 1

ftevent-42649f5c7677472f83eaa6cd4cde0dba Job enqueued.

➥Waiting for jobs ahead to complete

Możemy również co kilka sekund sprawdzać status zadania i uruchamiać w ten sposób kolejny przepływ pracy. W tym przypadku zadanie to działało przez około dwie godziny, zanim się zakończyło.

Do tego potrzebny jest pakiet IPython, który można zainstalować w Condzie za pomocą polecenia conda install ipython lub, jeśli korzystamy z Pipa, za pomocą polecenia – pip install ipython.

Listing 10: Sondowanie w celu sprawdzenia stanu zadania dostrajania

```
# Define the API version

API_VERSION = '2023-09-15-preview'

# Create an instance of the AzureOpenAI client

client = AzureOpenAI(

api_key=os.getenv('AOAI_FT_KEY'),

api_version=API_VERSION,

azure_endpoint = os.getenv('AOAI_FT_ENDPOINT'))

# Define the job ID of the fine-tuning job to track

JOB_ID = "ftjob-367ee1995af740a0bf24876221585f7a"

# Record the start time of the tracking

start_time = time.time()

# Get the status of the fine-tuning job

ft_job = client.fine_tuning.jobs.retrieve(JOB_ID)

status = ft_job.status

# If the job is not yet done, continue to poll its status every 30 seconds

while status not in ["succeeded", "failed"]:

ft_job = client.fine_tuning.jobs.retrieve(JOB_ID)

print(ft_job)

# Update the status

status = ft_job.status

# Print the elapsed time since the start of tracking

print("Elapsed time: {} minutes {} seconds".format(

➥int((time.time() - start_time) // 60),

➥int((time.time() - start_time) % 60)))

# Print the current status

print(f'Status: {status}')

# Clear the output before displaying new output – prevents flickering

clear_output(wait=True)
```

```
# Wait for 30 seconds before the next poll

time.sleep(30)

# Once the job is done, print its final status

print(f'Fine-tuning job {JOB_ID} finished with status: {status}')
```

W zależności od modelu i długości kolejki do zaplanowania zadania dostrajania, wykonanie jednego zadania dostrajania może zająć kilka godzin. W tym czasie otrzymujemy metryki szkoleniowe, które pomagają nam zrozumieć, jak przebiega proces szkolenia.

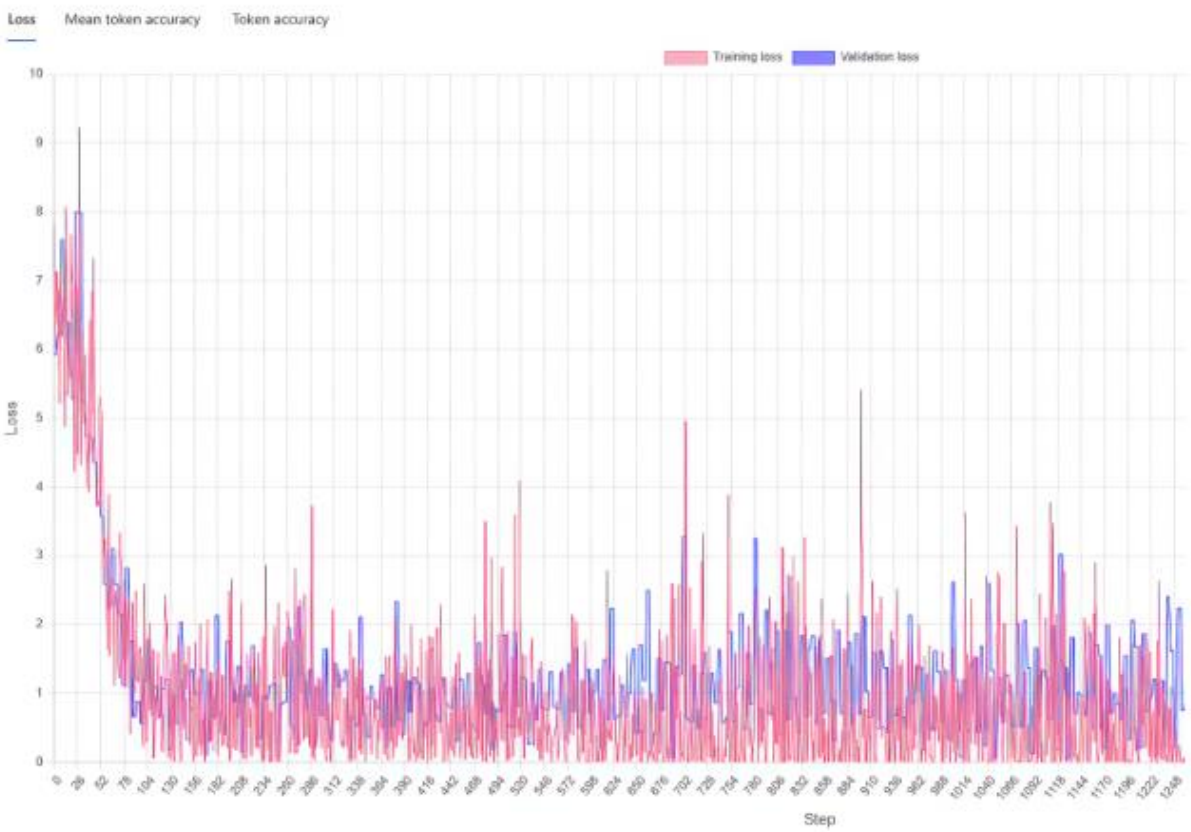
Dostrajanie metryk treningowych

Jak wspomniano wcześniej, trening może zająć kilka godzin; w przypadku bardziej złożonych i większych modeli może to zająć kilka dni. Trening w tym czasie nie jest czarną skrzynką; w trakcie procesu możemy uzyskać szczegółowe informacje na temat kluczowych metryk, aby uzyskać ogólny obraz tego, co się dzieje. Mamy trzy kluczowe metryki, które można śledzić: stratę treningową, średnią dokładność tokena i dokładność tokena.

STRATA

Istnieją dwa aspekty straty: strata treningowa i strata walidacyjna. Strata treningowa mierzy różnicę między przewidywaniami modelu a rzeczywistymi wynikami. Niższa strata oznacza, że model jest dokładniejszy i ma mniej błędów. Niższe wartości straty wskazują na lepszą wydajność modelu, sugerując, że przewidywania modelu są bliższe rzeczywistym danym. Jeśli dysponujemy zbiorem danych walidacyjnych (co jest wysoce zalecane), mamy również dodatkowe metryki, które pozwalają nam mierzyć wydajność modelu. Strata walidacji to metryka mierząca błąd modelu w zestawie walidacyjnym, czyli części zestawu danych przeznaczonej do oceny wydajności modelu na nowych lub nieznanych danych. Stratę walidacyjną oblicza się poprzez zsumowanie błędów dla każdego przykładu w zestawie walidacyjnym, używając tej samej funkcji kosztu, co strata treningowa. Stratę walidacyjną zazwyczaj mierzy się po każdej epoce, czyli pełnym przejściu przez zestaw treningowy.

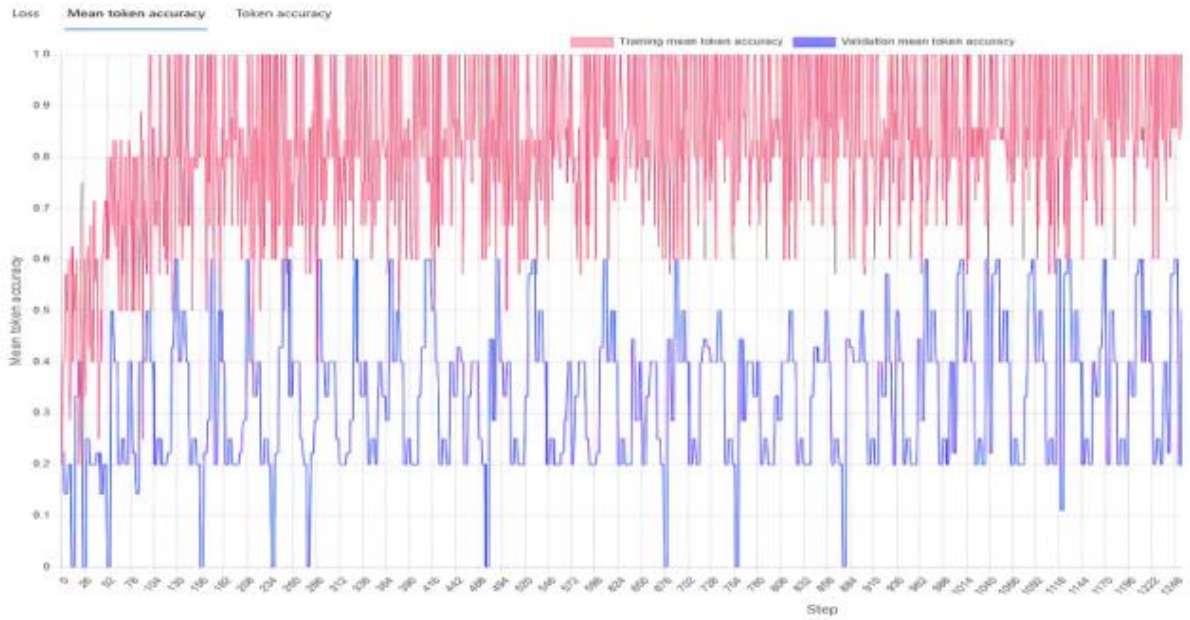
Rysunek 3 przedstawia przykład straty podczas dostrajania za pomocą Azure OpenAI oraz wydajność modelu podczas treningu.



Wykres na rysunku przedstawiający stratę treningową dla wyników dostrajania treningu ilustruje, jak dobrze model uczy się na podstawie danych treningowych. Widzimy wartość straty dla każdego kroku treningu, czyli partii przykładów treningowych. Oś x to numer kroku, a oś y to wartość straty. Wykres pokazuje, że strata maleje wraz z trenowaniem modelu na większej liczbie danych, co wskazuje na poprawę jego wydajności. Jednak strata nie osiąga zera, co oznacza, że model nadal ma pewne błędy i nie może idealnie dopasować danych. Jest to normalne, ponieważ nadmierne dopasowanie danych może prowadzić do słabej generalizacji nowych danych. Aby zinterpretować wykres i określić, czy model działa prawidłowo, a najlepiej, aby zapewnić dobre dopasowanie, chcemy, aby straty zarówno w procesie uczenia, jak i walidacji zmniejszyły się do poziomu stabilności z minimalną różnicą między nimi, co wskazuje na to, że model dobrze się uczy i generalizuje. Jeśli straty w procesie uczenia maleją, a straty w procesie walidacji rosną, model może nadmiernie dopasowywać dane treningowe i nie generalizować dobrze na nowe dane. Wreszcie, jeśli straty w procesie uczenia i walidacji pozostają wysokie, model może być niedopasowany, co oznacza, że nie uczy się wystarczająco dobrze podstawowych wzorców danych. Należy wziąć pod uwagę skalę strat i liczbę kroków treningowych. Model może wymagać dalszego trenowania, jeśli straty są nadal wysokie lub straty w procesie walidacji jeszcze się nie ustabilizowały. Dla osób z doświadczeniem lub wiedzą z zakresu modeli uczenia maszynowego, ogólne podejście do podziału zbiorów danych treningowych i walidacyjnych oraz interpretacji tych metryk jest bardzo podobne. Interesującym zjawiskiem jest to, że dane na wykresie strat fluktuują, co wskazuje, że wartość strat może się różnić w zależności od próbek w każdej partii. Zaszumiony model jest normalny; Jednak podczas dostrajania model uczy się i poprawia swoją wydajność, o ile strata maleje z czasem. Aby sprawdzić, czy dostrajanie jest dobre, zazwyczaj szukamy niskiej i stabilnej straty walidacyjnej zbliżonej do straty szkoleniowej. Progi dla wartości uznawanych za dobre straty są subiektywne i będą się różnić w zależności od złożoności zadania i charakteru danych.

ŚREDNIA DOKŁADNOŚĆ TOKENÓW

Średnia dokładność tokenów mierzy, jak dobrze dostrojony model poprawnie przewiduje każdy token w sekwencji wyjściowej, którą generuje lub przewiduje podczas uczenia. Jest ona wyrażana w procentach, czyli jako odsetek tokenów, które model poprawnie przewiduje w zbiorze danych. Na przykład, jeśli średnia dokładność tokenów wynosi 90%, oznacza to, że średnio model poprawnie przewiduje 90% tokenów. Jest to średnia obliczona poprzez podzielenie liczby poprawnie przewidzianych tokenów przez całkowitą liczbę tokenów w danych wyjściowych. Podobnie jak w przypadku straty średniej dokładności tokenów, mamy dwie metryki: jedną dla treningu, a drugą dla walidacji (zakładając, że dostarczono zbiór danych walidacyjnych). Rysunek 4 przedstawia średnią dokładność tokenów w procesie dostrajania dla treningu i walidacji.

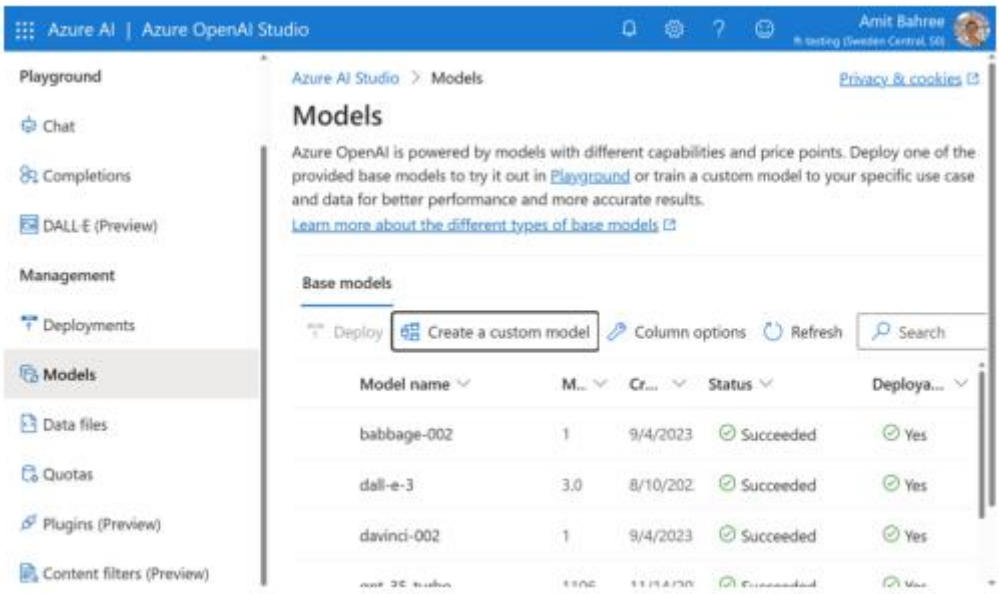


Średnia dokładność tokenów dla treningu to średnia dokładność przewidywań modelu na podstawie danych treningowych. Mierzy ona, jak dobrze model uczy się na podstawie danych treningowych i adaptuje się do nich. Wysoka średnia dokładność tokenów dla treningu sugeruje, że model uczy się efektywnie na podstawie danych treningowych. Natomiast średnia dokładność tokenów dla walidacji to średnia dokładność przewidywań modelu na podstawie danych walidacyjnych. Mierzy ona, jak dobrze model generalizuje na nowe dane, których wcześniej nie widział. Wysoka średnia dokładność tokenów dla walidacji sugeruje, że model nie przeucza danych treningowych i może dobrze generalizować na nowe dane. Różnica między tymi dwiema metrykami może pomóc w identyfikacji, czy model nie przeucza danych treningowych. Załóżmy, że średnia dokładność tokenów dla treningu jest znacznie wyższa niż średnia dokładność tokenów dla walidacji. W takim przypadku sugeruje to, że model nadmiernie dopasowuje się do danych treningowych i nie generalizuje dobrze na nowe dane. Natomiast jeśli średnia dokładność tokenów walidacyjnych jest znacznie niższa niż średnia dokładność tokenów treningowych, sugeruje to, że model nie dopasowuje się do danych treningowych i nie uczy się skutecznie. Ta metryka jest przydatna do oceny wydajności precyzyjnie dostrojonego modelu na danych treningowych. Dobra średnia dokładność tokenów może być względna i zależy od konkretnego zadania lub aplikacji. Zasadniczo wyższa wartość (bliższa 1,0) oznacza lepszą wydajność. Nie odzwierciedla ona jednak, jak dobrze model generalizuje na nowe lub nieznane dane. Należy pamiętać, że interpretacja tych metryk może zależeć od konkretnego zadania lub aplikacji. Dlatego ważne jest, aby rozważyć inne metryki i oceny jakościowe, aby uzyskać kompleksowy obraz wydajności modelu. Jakość średniej dokładności tokenów zależy od złożoności zadania i charakteru tekstu. Wyższej dokładności (bliższej 100%) oczekuje się w przypadku prostszych zadań lub tekstów o przewidywalnych

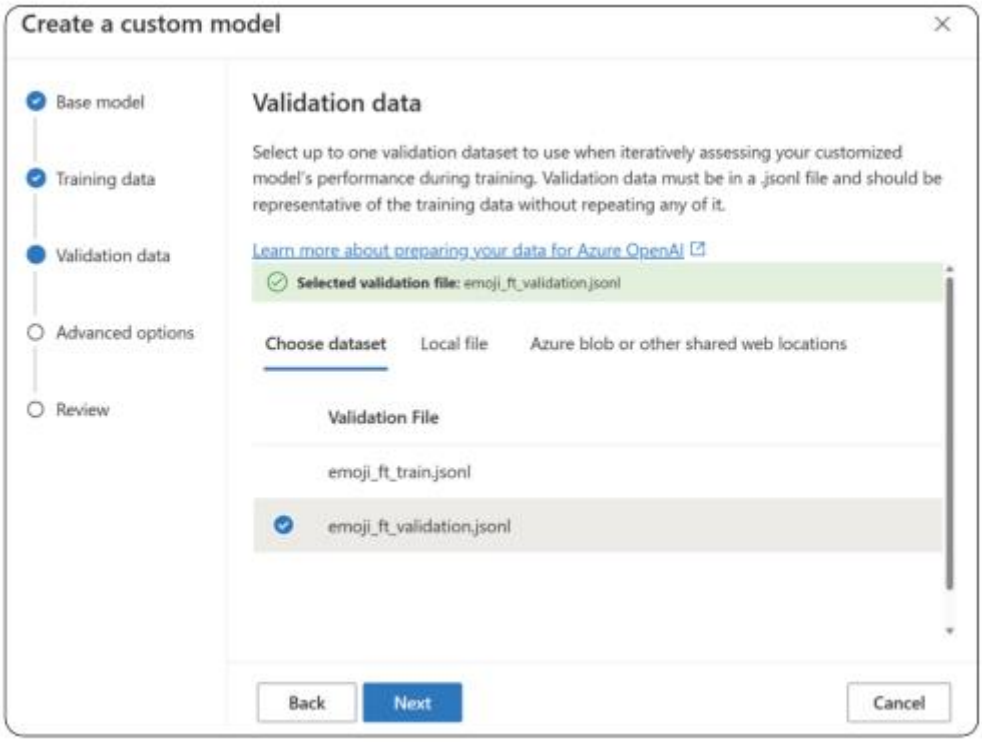
wzorcach. Niższa dokładność może być nadal korzystna w przypadku bardziej złożonych zadań lub zróżnicowanych tekstów. Jednym ze sposobów oceny, czy średnia dokładność tokenów jest dobra, jest porównanie jej z wartością bazową lub z wydajnością innych modeli dla tego samego zadania. Jeśli dokładność Twojego modelu jest wyższa niż modelu bazowego lub podobnych modeli, to dobry znak. Teraz, gdy rozumiemy podstawowe zasady dostrajania i korzystania z interfejsu wiersza poleceń (CLI) lub kodu, przyjrzyjmy się, jak możemy to osiągnąć za pomocą Azure OpenAI i graficznego interfejsu użytkownika. Jak wspomniano wcześniej, posłużymy się Azure OpenAI jako przykładem, ale ten sam proces dotyczy OpenAI.

Dostrajanie za pomocą Azure OpenAI

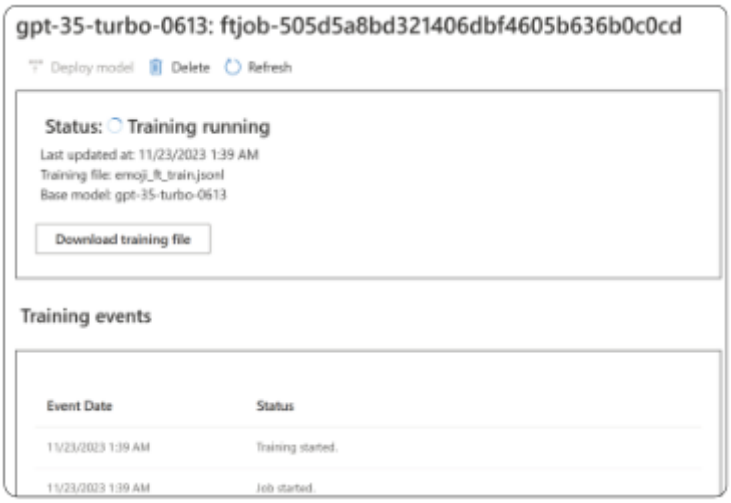
Zamiast korzystać z zestawu SDK i interfejsu wiersza poleceń, mamy również interfejs wizualny, który możemy wykorzystać do osiągnięcia tego samego rezultatu. Często ręczne wykonanie tej czynności jest lepszym rozwiązaniem niż użycie kodu. Aby rozpocząć dostrajanie w Azure OpenAI, po zalogowaniu się do Azure Portal i w AI Studio, w obszarze modeli, wybieramy opcję utworzenia modelu niestandardowego (rysunek 5).



Przechodzimy przez kreator i wybieramy przesłanie zestawów danych treningowych i walidacyjnych, jak pokazano na rysunku 6.



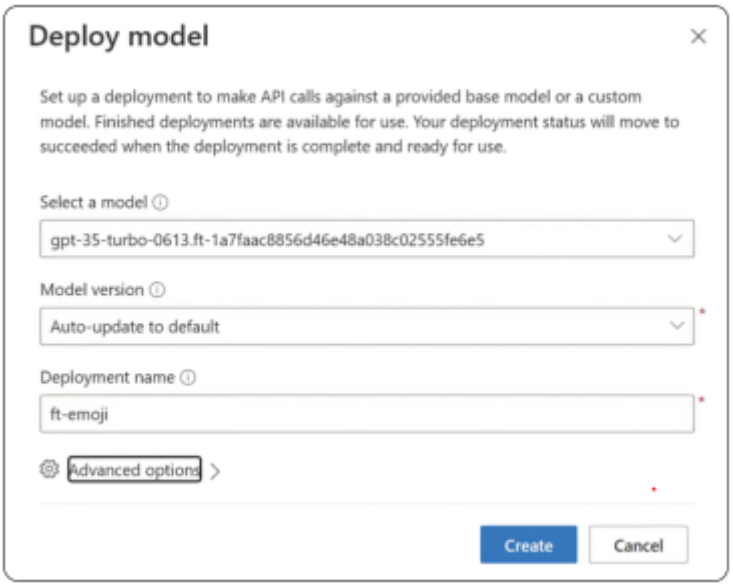
Uwaga: jeśli zostały one przesłane za pomocą zestawu SDK, znajdziemy je tutaj, o ile znajdują się w tych samych dzierżawach i mają to samo wdrożenie w punkcie końcowym. Rysunek 7 przedstawia status i szczegóły każdego z naszych zadań szkoleniowych.



Mając już dostrojony model, musimy go wdrożyć w środowisku testowym, aby przeprowadzić ocenę.

Wdrożenie dostrojonego modelu

Wdrożenie dostrojonego modelu jest dość proste. Nowy, dostrojony model pojawia się jako kolejny model dostępny do użycia w naszej dzierżawie Azure lub subskrypcji OpenAI, jak pokazano odpowiednio na rysunkach 8 i 9.

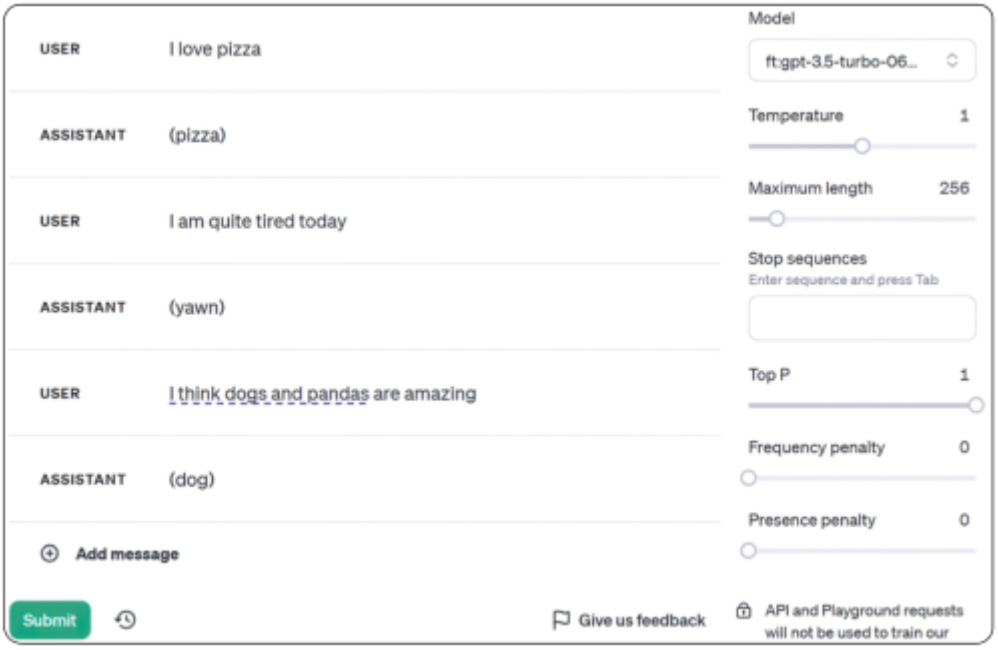


Firma OpenAI uruchomiła w środowisku roboczym funkcję, która pozwala użytkownikom zobaczyć, jak dopracowany model różni się od modelu bazowego, co może być przydatne wizualnie, ale nieefektywne.

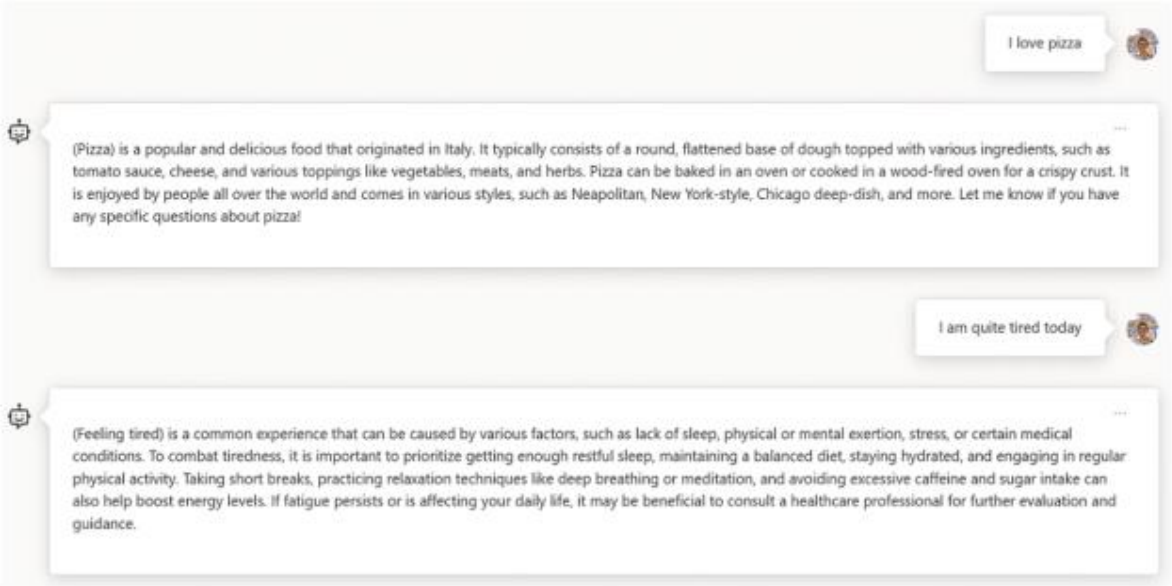
Wnioskowanie: Dopracowany model

Wracając do naszego zadania, mamy teraz dopracowany model dla EmojiBot, w którym bot odpowiada emotikonami w formacie używanym przez Microsoft Teams. Rysunek 10 pokazuje, jak zachowuje się gotowy model GPT-3.5 Turbo, gdy zostanie poproszony o odpowiedź emotikonami; jest to oczekiwane, ale nie będzie działać w Teams.

Jednak doświadczenie z tymi samymi pytaniami za pomocą naszego udoskonalonego, zaawansowanego EmojiBota jest zupełnie inne, jak pokazano na rysunku 11. Tutaj, na te same pytania co poprzednio, otrzymujemy odpowiedź w formacie, którego będziemy mogli używać w Teams.



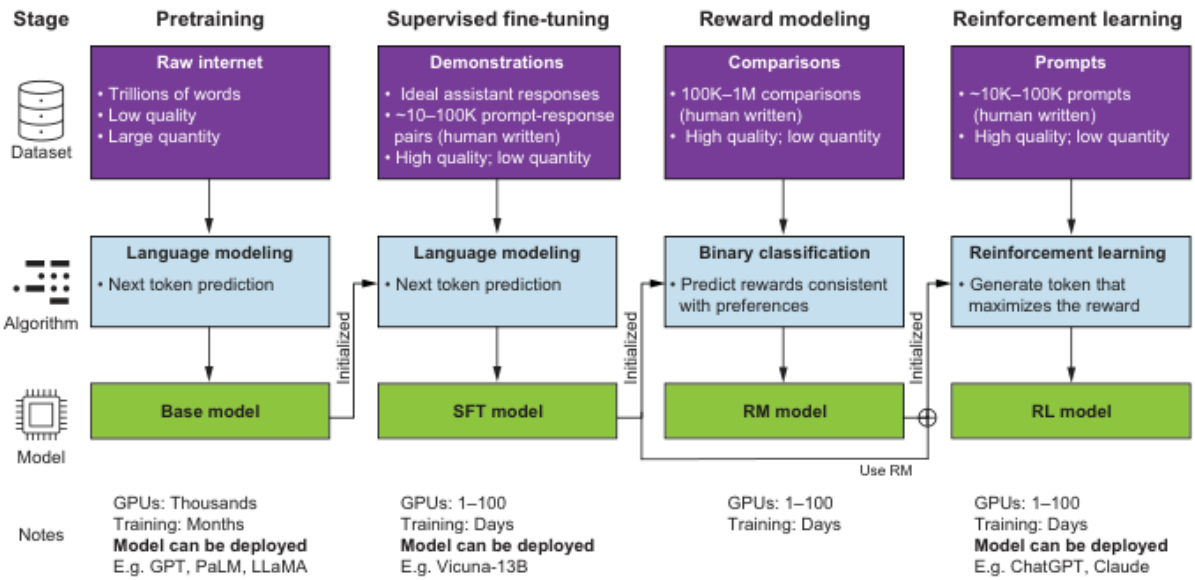
Jednak łatwo jest uzyskać całkowicie niepoprawne wyniki w odpowiedzi na te same pytania z poprzednich pytań i przy tych samych ustawieniach parametrów (rysunek 12). Dopracowaną odpowiedź modelu widzimy w emotikonach – (Pizza) i (Czuję się zmęczony) – ale wynik nie jest taki, jakiego oczekiwaliśmy.



Aby rozwiązać ten problem, musimy zmodyfikować komunikat systemowy, aby model reagował za pomocą emoji, tam gdzie to możliwe. To doskonały sposób na zakończenie, przypominając, że złożone podejście obejmujące inżynierię natychmiastową, RAG i dostrajanie (tam, gdzie wymaga tego zadanie) jest właściwym podejściem w prawie wszystkich przypadkach. Teraz, gdy poznaliśmy już metody dostrajania modelu i kroki, które należy podjąć, przełączmy się i przyjrzyjmy się niektórym podstawom technologii, która umożliwi to działanie. Ściśle rzecz biorąc, nie jest to wymagane do dostrajania, ale pomoże nam zrozumieć pewne niuanse, aby osiągnąć lepsze rezultaty w dostrajaniu. Zaczniemy od zrozumienia, jak szkolimy LLM i, ogólnie rzecz biorąc, na czym polegają poszczególne kroki.

Szkolenie LLM

Do zrozumienia adaptacji modelu oraz technik i związanych z nimi ograniczeń pomocne będzie przeanalizowanie, co to oznacza i co jest potrzebne do przeprowadzenia pełnego szkolenia LLM. Ogólnie rzecz biorąc, gdybyśmy mieli przeprowadzić pełne szkolenie i zbudować LLM od podstaw, szkolenie to obejmowałoby cztery główne etapy, jak pokazano na rysunku 13.



Przyjrzyjmy się każdemu etapowi bardziej szczegółowo.

Wstępne trenowanie

Podstawowe modele LLM są tworzone na tym wstępnym etapie. Omówiliśmy je w rozdziale 2. Są to oryginalne, wstępnie wytrenowane modele, trenowane na ogromnym korpusie danych tekstowych. Potrafią generować tekst na podstawie wzorców poznanych podczas trenowania. Niektórzy nazywają je również surowymi modelami językowymi.

UWAGA: Choć wydajne, te modele bazowe są mniej odpowiednie do zastosowań ogólnych, ponieważ mogą wymagać dostosowania swoich odpowiedzi do konkretnych intencji lub instrukcji użytkownika. Przypominają raczej surowe silniki generowania tekstu, pozbawione udoskonalonej zdolności do rozumienia i przestrzegania niuansów poleceń użytkownika. Modele bazowe nie odpowiadają na pytania i często odpowiadają za pomocą większej liczby pytań. Natomiast instruktorzy są dostosowywani tak, aby byli bardziej interaktywni i przyjaźni dla użytkownika, co czyni ich bardziej odpowiednimi do szerokiego zakresu zastosowań, od chatbotów obsługi klienta po narzędzia edukacyjne, gdzie zrozumienie i dokładne przestrzeganie instrukcji ma kluczowe znaczenie.

Dostrajanie nadzorowane: Dostrajanie nadzorowane (SFT) to kolejny etap. Na tym etapie model bazowy jest udoskonalany za pomocą wysokiej jakości danych specyficznych dla danej dziedziny. Te zbiory danych składają się z par poleceń i odpowiedzi, tworzonych ręcznie (często przez ludzkich wykonawców), których jest mniej niż na poprzednim etapie, ale są one znacznie lepszej jakości. Kontrahenci postępują zgodnie ze szczegółową dokumentacją, aby tworzyć pary monit-odpowiedź, zapewniając trafność i jakość. Podobnie jak na ostatnim etapie wstępnego szkolenia, model SFT jest trenowany w celu przewidywania kolejnego tokena w tych parach, ale są one mniej dokładne i kontekstowe podczas generowania odpowiedzi.

SFT to technika optymalizacji modeli LLM na oznaczonych danych dla konkretnego zadania downstream, takiego jak analiza sentymentu, podsumowanie tekstu lub tłumaczenie maszynowe. W dalszej części rozdziału omówimy dodatkowe szczegóły dotyczące metod i podejść SFT.

Modelowanie nagród: Trzecią fazą jest modelowanie nagród, pierwsza część procesu uczenia się wzmacniającego na podstawie informacji zwrotnej od człowieka (RLHF). Głównym celem na tym etapie jest opracowanie modelu, który może oceniać i klasyfikować odpowiedzi na podstawie ich jakości i trafności. W tym celu model SFT (z poprzedniego etapu) generuje wiele odpowiedzi na monit, które następnie kontrahenci klasyfikują na podstawie różnych kryteriów, takich jak wiedza specjalistyczna, weryfikacja faktów i wykonanie kodu. Te rankingi trenują model nagrody, który uczy się punktować odpowiedzi, tak jak robią to ludzie.

Uczenie przez wzmacnianie: To druga część procesu RLHF, której celem jest zwiększenie zdolności modelu językowego do generowania wysokiej jakości odpowiedzi poprzez iteracyjne sprzężenie zwrotne. Na tym ostatnim etapie model nagrody punktuje odpowiedzi generowane przez model SFT dla wielu podpowiedzi. Punkty te są wykorzystywane do dalszego trenowania modelu SFT, co ostatecznie prowadzi do stworzenia modelu RLHF. Model RLHF dostosowuje modele LLM do ludzkich preferencji lub oczekiwań dotyczących danego zadania lub dziedziny, takiej jak czat, kodowanie czy pisanie kreatywne.

Bezpośrednia optymalizacja polityki

Bezpośrednia optymalizacja polityki (DPO) [6] to kolejna technika, będąca nowym rodzajem parametryzacji modelu nagrody w RLHF, wykorzystywana do precyzyjnego dostrajania modeli LLM w celu dopasowania ich do naszych preferencji. Wykorzystuje ona zależność między funkcjami nagrody

a optymalnymi politykami. Pozwala nam pominąć opisany wcześniej etap modelowania nagrody, o ile ludzkie sprzężenie zwrotne można wyrazić w kategoriach binarnych — czyli dokonać wyboru między dwiema opcjami. DPO może rozwiązać problem maksymalizacji nagrody z ograniczeniami w pojedynczej fazie szkolenia polityki, traktując go zasadniczo jako problem klasyfikacji. PPO wymaga modelu nagrody i złożonego procesu optymalizacji opartego na RL; DPO pomija jednak etap modelowania nagrody i bezpośrednio optymalizuje model języka na podstawie danych o preferencjach, co może być prostsze i bardziej wydajne. Ponieważ DPO eliminuje potrzebę trenowania modelu nagrody zamiast trenowania modelu nagrody i optymalizacji polityki na podstawie tego modelu, możemy bezpośrednio optymalizować politykę. Ta cecha sprawia, że podejście to jest szybsze i wymaga użycia mniejszej ilości zasobów niż w przypadku RLHF z PPO.

Techniki adaptacji modelu

Istnieje kilka technik adaptacji modelu, z których każda zapewnia unikalne podejście i jest odpowiednia dla różnych scenariuszy, w zależności od konkretnych wymagań (tj. rozmiaru modelu, dostępnych zasobów obliczeniowych i pożądanego poziomu adaptacji). Jedną z głównych technik powszechnie stosowanych do adaptacji modeli LLM jest algorytm niskiego rankingu adaptacyjnego (LoRA), który zostanie omówiony bardziej szczegółowo w następnym rozdziale. W LoRA, zamiast aktualizować wszystkie wagi w modelu, modyfikowany jest tylko niewielki podzbiór parametrów, wprowadzany jako macierze niskiego rzędu. Takie podejście umożliwia efektywne trenowanie i adaptację, przy jednoczesnym zachowaniu większości struktury i wiedzy wstępnie wytrenowanego modelu. Efektywne dostrajanie parametrów (PEFT) to koncepcja w uczeniu maszynowym, która odnosi się do metod adaptacji i dostrajania dużych, wstępnie wytrenowanych modeli, takich jak GPT-3.5, w celu zminimalizowania liczby parametrów wymagających aktualizacji. To podejście jest szczególnie cenne w przypadku dużych modeli, ponieważ zmniejsza wymagania obliczeniowe i może łagodzić problemy, takie jak nadmierne dopasowanie. Techniki PEFT mają na celu uczynienie dostrajania bardziej dostępnym i wydajnym, zwłaszcza dla użytkowników o ograniczonych zasobach obliczeniowych — LoRA jest przykładem metody PEFT. Katastroficzne zapominanie to zjawisko, w którym model traci zdolność do dobrego wykonywania poprzednich zadań po dostrojeniu do nowych zadań. Może się to zdarzyć, gdy model nadpisuje swoje oryginalne parametry parametrami specyficznymi dla zadania, zapominając w ten sposób ogólną wiedzę, której nauczył się podczas wstępnego trenowania. Podczas wdrażania PEFT w celu zapobiegania katastrofalnemu zapominaniu dostrajamy tylko mały podzbiór parametrów, jednocześnie utrzymując większość wstępnie wytrenowanych parametrów na stałym poziomie. W ten sposób model może zachować swoją zdolność generalizacji i adaptacji do nowych zadań bez utraty poprzedniej wydajności. Nadzorowane dostrajanie (SFT) to inny rodzaj techniki adaptacji; Jest to specyficzny rodzaj dostrajania, w którym model jest dodatkowo trenowany na oznaczonym zbiorze danych. Jest nadzorowany, ponieważ proces trenowania wykorzystuje zbiór danych, który paruje dane wejściowe z odpowiednimi danymi wyjściowymi (etykietami). SFT jest szczególnie powszechne w zadaniach takich jak klasyfikacja, gdzie model musi nauczyć się kojarzenia określonych danych wejściowych z oznaczonymi danymi wyjściowymi. SFT jest podzbiorem szerszego procesu dostrajania, specjalnie dostosowanym do sytuacji, w których dostępne są oznaczone dane. Oczywiście każda technika ma swoje unikalne podejście i jest odpowiednia dla różnych scenariuszy, w zależności od konkretnych wymagań, takich jak rozmiar modelu, dostępne zasoby obliczeniowe i pożądany poziom adaptacji. Tabela 2 przedstawia niektóre z ważniejszych, oprócz LoRA.

Technika: Opis

Dostrajanie za pomocą komend: Dostrajanie za pomocą komend to technika dostosowywania LLMS do różnych zadań poprzez dostarczanie określonych wskazówek lub komend, które kierują ich generowaniem lub przewidywaniem. Nie wymaga ponownego trenowania modelu ani aktualizacji jego

wag, co czyni ją szybszą i tańszą niż dostrajanie precyzyjne. Jest szczególnie przydatna w zadaniach wymagających jedynie niewielkiej adaptacji.

Moduły adaptera: Modele adaptera są używane w dostrajaniu LLM do dodawania małych i specyficznych dla danego zadania modułów (małych sieci neuronowych) do wstępnie wytrenowanego modelu i trenowania tylko tych modułów na danych specyficznych dla danego zadania. Są one również elastyczne i modułowe, ponieważ można je łatwo dodawać lub usuwać dla różnych zadań bez wpływu na wstępnie wytrenowany model.

Tylko odchylenie (BitFit): Tylko odchylenie (BitFit) to technika dostrajania LLM poprzez modyfikację tylko warunków odchylenia modelu lub ich podzbioru. Oferuje minimalistyczne podejście do adaptacji, wymagając jeszcze mniej parametrów do trenowania niż LoRA. BitFit opiera się na dostrajaniu, głównie eksponując wiedzę zdobytą podczas wstępnego treningu, a nie nabywając nowej wiedzy specyficznej dla danego zadania.

Zamrażanie warstw: Zamrażanie warstw to technika dostrajania, która utrzymuje niektóre warstwy modelu niezmienne, a aktualizuje tylko pozostałe. Ta metoda pozwala na większą kontrolę nad tym, które aspekty modelu są dostosowywane i może skrócić czas treningu.

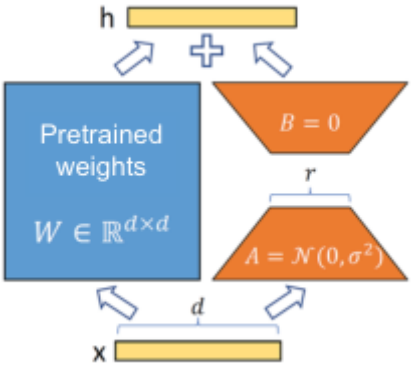
Destylacja wiedzy: Polega na trenowaniu mniejszego, bardziej wydajnego modelu (ucznia), aby naśladował zachowanie większego, wstępnie wytrenowanego modelu (nauczyciela). Ta metoda jest przydatna do wdrażania LLM w środowiskach o ograniczonych zasobach.

Meta-uczenie: Koncentruje się na trenowaniu modelu, aby szybko uczył się nowych zadań przy minimalnej ilości danych; często obejmuje trenowanie różnych zadań, aby model mógł efektywniej dostosowywać się do nowych, nieznanych zadań. Dostrajanie różnicowe prywatności: Wykorzystuje techniki ochrony prywatności podczas dostrajania w celu ochrony poufnych danych. Jest to niezbędne w aplikacjach, w których prywatność i bezpieczeństwo danych mają kluczowe znaczenie. Uczenie się przez wzmacnianie na podstawie informacji zwrotnej od człowieka (RLHF): Polega na dostrajaniu modeli w oparciu o informacje zwrotne lub nagrody uzyskane w wyniku interakcji lub ocen z ludźmi. Jest przydatne w zadaniach, w których osąd człowieka ma kluczowe znaczenie, takich jak moderacja treści.

Każda technika wiąże się z kompromisem między wymaganymi zasobami obliczeniowymi, osiągniętym poziomem specjalizacji oraz zachowaniem pierwotnych możliwości modelu. Wybór techniki zależy od konkretnego zastosowania, ograniczeń środowiska wdrożeniowego oraz celów adaptacji modelu. Teraz, gdy znasz już różne techniki, przyjrzyjmy się bliżej LoRA, głównej technice dostrajania dużych modeli, takich jak GPT.

Adaptacja niskiej rangi

LoRA, czyli adaptacja niskiej rangi , to metoda zaprojektowana specjalnie do adaptacji modeli LLM. Stanowi ona efektywną alternatywę dla tradycyjnych metod dostrajania, co jest szczególnie przydatne w scenariuszach, w których dostrajanie dużych modeli może być zasobochłonne i trudne. LoRA opiera się na wprowadzaniu minimalnych, ale strategicznych modyfikacji do wstępnie wytrenowanego modelu bez zmiany jego całej architektury. Osiąga to poprzez wprowadzenie koncepcji macierzy niskiej rangi. Zamiast modyfikować całe macierze wag sieci neuronowej, LoRA wstawia do modelu małe macierze niskiej rangi. Macierze te są nakładane na warstwy modelu (uwagi i sprzężenia zwrotnego) podczas przebiegów w przód i w tył, jak pokazano na rysunku 14.



Jak widzieliśmy, modele LLM są zbudowane w oparciu o architektury głębokiego uczenia składające się z wielu warstw zaprojektowanych do przetwarzania i rozumienia języka ludzkiego. Ponadto LoRA ponownie trenuje selektywnie tylko te macierze niskiej rangi, podczas gdy oryginalne, wstępnie wytrenowane wagi pozostają zamrożone. To selektywne ponowne trenowanie znacząco zmniejsza zapotrzebowanie na zasoby obliczeniowe. Na rysunku 14, wejście (X) na dole diagramu reprezentuje dane wejściowe wprowadzone do warstwy z wstępnie wytrenowanymi wagami. A i B to parametry adaptacyjne, które zostaną zaktualizowane podczas dostrajania, a W to oryginalna, wstępnie wytrenowana waga, która pozostaje zamrożona. Chcąc dostroić zadanie, możemy przechowywać i ładować tylko kilka parametrów specyficznych dla zadania wraz z wstępnie wytrenowanym modelem. Takie podejście pomaga poprawić wydajność w czasie wykonywania dla różnych późniejszych adaptacji. Daje to LoRA kilka korzyści, w tym:

* Efektywność zasobów — LoRA wymaga znacznie mniejszej mocy obliczeniowej niż tradycyjne dostrajanie pełnego modelu, co czyni ją bardziej dostępną do adaptacji dużych modeli.

* Zachowanie generalizacji — LoRA zachowuje możliwości generalizacyjne bazowego modelu LLM, nie zmieniając oryginalnych, wstępnie wytrenowanych wag, jednocześnie umożliwiając specjalizację.

* Szybsza adaptacja — proces jest szybszy dzięki mniejszej liczbie zaktualizowanych parametrów, co umożliwia szybkie wdrażanie zaadaptowanych modeli.

* Skalowalność — LoRA jest szczególnie skuteczna w przypadku dużych modeli, w których dostrajanie całego modelu może być niepraktyczne ze względu na ograniczenia zasobów.

LoRA to ekonomiczna i wydajna metoda dla modeli językowych, która umożliwia szybkie przełączanie się między zadaniami. QLoRA to wariant LoRA, który dodatkowo redukuje liczbę parametrów poprzez kwantyzację macierzy niskiego rzędu i może osiągnąć nawet 99% redukcję parametrów (poprzez implementację 8-bitowego optymalizatora do kwantyzacji), przy jednoczesnym zachowaniu lub poprawie wydajności modelu.

KWANTYZACJA

Kwantyzacja to kolejna technika, która zmniejsza zapotrzebowanie modelu na pamięć i moc obliczeniową poprzez reprezentowanie parametrów za pomocą mniejszej liczby bitów. Kwantyzacja modelu oznacza zmniejszenie precyzji parametrów modelu, takich jak wagi i odchylenia, z liczb zmiennoprzecinkowych o wysokiej precyzji (32- lub 16-bitowych) do liczb o niskiej precyzji (8- lub 4-bitowych). Może to zmniejszyć rozmiar modelu i przyspieszyć wnioskowanie, ale może również wpłynąć na dokładność modelu. Kwantowanie jest szczególnie przydatne w modelach LLM, które mogą mieć miliardy parametrów i wymagać dużej ilości pamięci oraz obliczeń. Kwantowanie modelu pozwala na bardziej wydajne i skalowalne wdrażanie i wnioskowanie. Na przykład DistilBERT to skwantowana

wersja BERT, modelu LLM dla przetwarzania języka naturalnego. Ma o 40% mniej parametrów niż BERT, ale zachowuje 97% wydajności BERT. Na pierwszy rzut oka kwantyzacja jest podobna do LoRA, ponieważ oba modele mają na celu poprawę wydajności i skalowalności modeli LLM. Jednak różnią się one znacznie pod względem podejścia i kompromisów:

* LoRA zmniejsza liczbę parametrów możliwych do trenowania poprzez zamrożenie wstępnie wytrenowanych wag modelu i wstrzyknięcie macierzy niskiego rzędu do każdej warstwy. Pozwala to na szybsze dostrajanie i adaptację do nowych zadań. LoRA zachowuje również pełną precyzję wag modelu, co oznacza, że nie zmniejsza zużycia pamięci ani opóźnień wnioskowania.

* Kwantowanie zmniejsza zapotrzebowanie modelu na pamięć i obliczenia poprzez reprezentowanie parametrów za pomocą mniejszej liczby bitów, takich jak INT4. Ta cecha pozwala na mniejsze rozmiary modelu i szybsze wnioskowanie, ale wprowadza również błędy kwantyzacji i szum, które mogą obniżyć wydajność modelu. Kwantowanie wymaga również starannej kalibracji i optymalizacji w celu zminimalizowania utraty dokładności i odporności.

* Kwantowane LoRA (QLoRA) to kolejna technika, której celem jest poprawa wydajności parametrów precyzyjnie dostrojonych modeli LLM. Rozszerza ona LoRA poprzez dodanie kwantyzacji do procesu. Oznacza to, że wagi adapterów LoRA są kwantyzowane z niższą precyzją, np. 4 bitami, co znacznie zmniejsza rozmiar pamięci modelu. Główną zaletą QLoRA jest jej zdolność do równoważenia wydajności i efektywności pamięci, co czyni ją odpowiednią opcją w przypadku ograniczonych zasobów. Pomimo niższej precyzji, QLoRA, w różnych zadaniach, wykazuje podobny poziom efektywności jak jej niekwantyzowana wersja, LoRA. To sprawia, że QLoRA jest atrakcyjnym wyborem dla użytkowników, którzy chcą korzystać z wydajnych modeli językowych bez wysokich kosztów obliczeniowych.

Główne kompromisy między LoRA i QLoRA dotyczą równowagi między wydajnością, efektywnością pamięci i zasobami obliczeniowymi. LoRA zapewnia dobrą równowagę między wydajnością a efektywnością, podczas gdy QLoRA maksymalizuje oszczędność pamięci, co może mieć kluczowe znaczenie w niektórych przypadkach użycia. Wybór między nimi zależy od specyficznych potrzeb danego zadania i ograniczeń środowiska wdrożenia. Eksperymentowanie jest ważne, aby określić, która metoda najlepiej odpowiada potrzebom. Jeśli wysoka dokładność jest bardzo ważna, a zasoby obliczeniowe są wystarczające, LoRA może być najlepszym wyborem. Jeśli ważniejsza jest efektywność pamięci, QLoRA będzie lepszym wyborem, zwłaszcza jeśli niewielki spadek wydajności jest akceptowalny dla danej aplikacji.

Nauczanie nowej wiedzy za pomocą dostrajania

Często panuje błędne przekonanie, że dostrajanie uczy model nowej wiedzy (lub informacji). To nieprawda. SFT nie uczy modelu nowej wiedzy w tradycyjnym sensie. Wiedza dostrojonego modelu ogranicza się do tego, co było obecne w danych przedtreningowych do ostatniej aktualizacji. Model nie nabywa nowej wiedzy zewnętrznej podczas procesu dostrajania. Zamiast tego udoskonala i dostosowuje istniejącą wiedzę i możliwości modelu, aby lepiej radził sobie z określonymi zadaniami lub w określonych domenach. SFT jedynie udoskonala istniejącą wiedzę, jak opisano tutaj:

* Wykorzystanie wiedzy wstępnie wytrenowanej — w SFT model został już wytrenowany na dużym, zróżnicowanym zbiorze danych. To wstępne szkolenie zapewnia szeroką bazę wiedzy ogólnej i rozumienia języka.

* Koncentracja na konkretnych zadaniach — podczas SFT istniejąca wiedza modelu jest udoskonalana, aby była bardziej efektywna w określonych zadaniach. Na przykład, jeśli dostosujesz model językowy

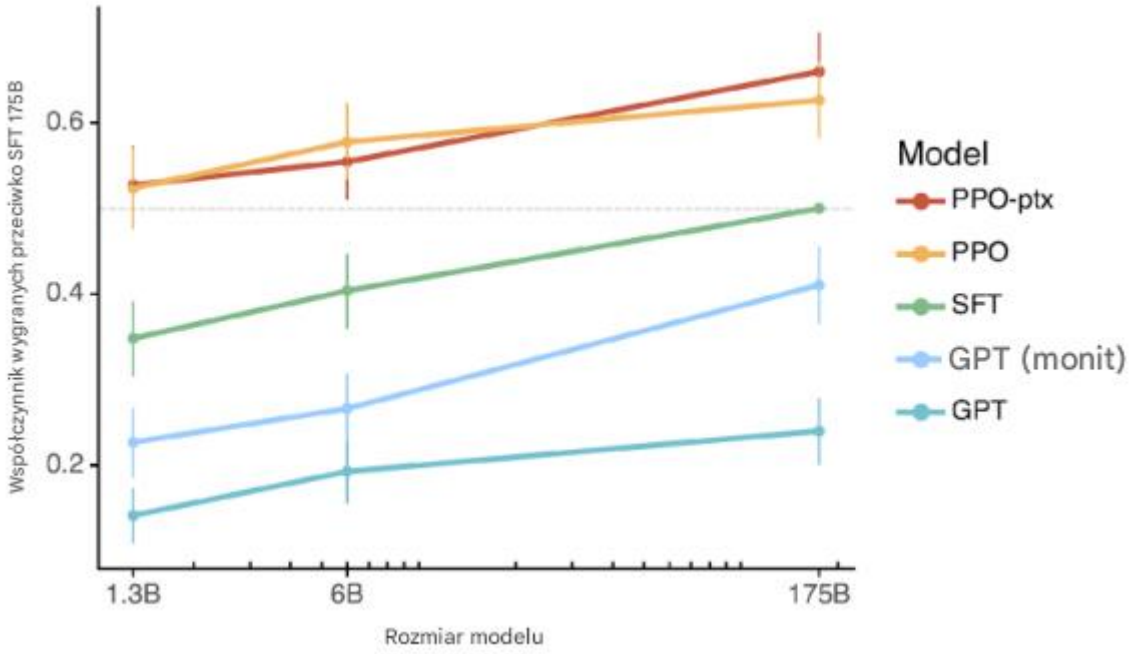
do tekstów medycznych, model ten staje się bardziej biegły w rozumieniu i generowaniu języka związanego z medycyną. Niekoniecznie jednak uczy on nowych faktów na temat medycyny.

* Dostosowywanie wag pod kątem trafności — dostrajanie skutecznie dostosowuje wewnętrzne parametry (lub wagi) modelu, aby uwydatnić pewne cechy lub wzorce podczas formułowania prognoz lub generowania tekstu. Ten proces sprawia, że model jest bardziej wrażliwy na niuanse konkretnych danych, na podstawie których został dostrojony.

SFT dostosowuje istniejącą wiedzę i możliwości modelu, aby były bardziej efektywne w określonych zastosowaniach, zamiast uczyć go nowej, zewnętrznej wiedzy. Proces ten polega na dostosowaniu wewnętrznego rozumienia modelu i mechanizmów generowania odpowiedzi, aby lepiej odpowiadały charakterystyce danych dostrajających.

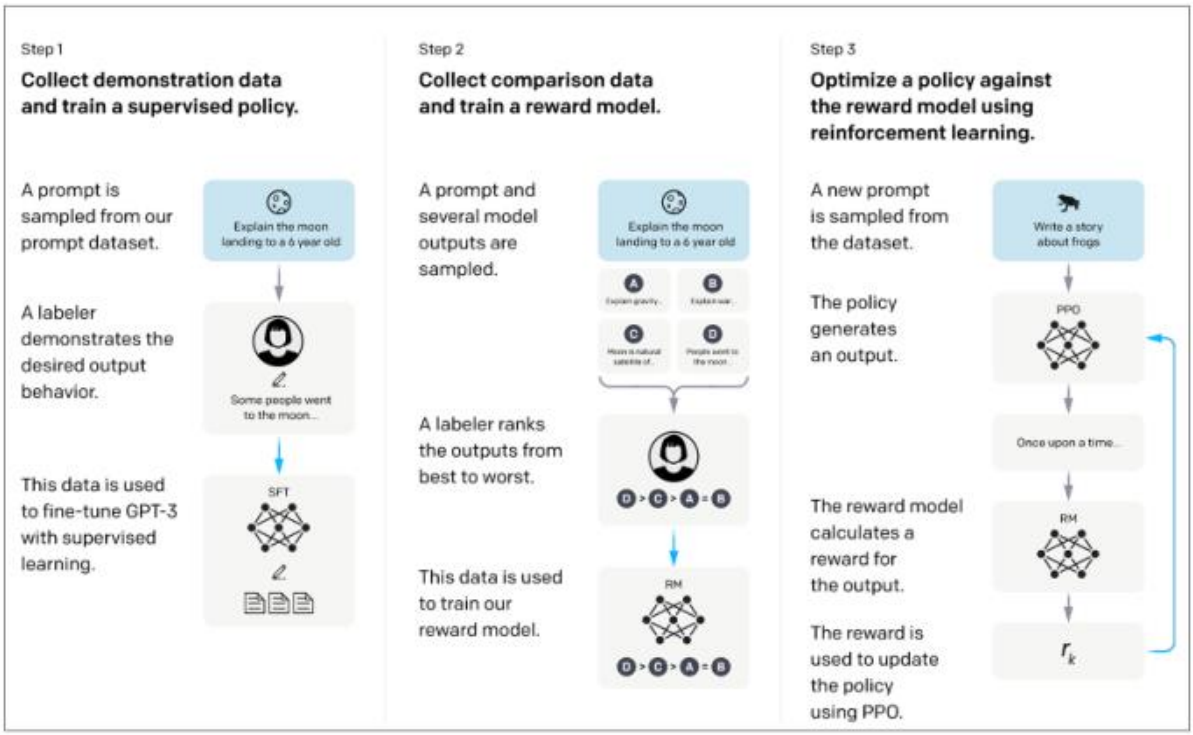
Przegląd RLHF

Uczenie się przez wzmacnianie na podstawie informacji zwrotnej od człowieka (RLHF) to zaawansowane podejście ML, które łączy uczenie przez wzmacnianie (RL) i uczenie nadzorowane. W przeciwieństwie do tradycyjnego RL, które opiera się na predefiniowanych funkcjach nagrody, RLHF integruje ludzki osąd z uczeniem się, prosząc ludzi o ocenę zachowania agenta i udzielenie informacji zwrotnej, takiej jak oceny, preferencje lub sugestie. Ta informacja zwrotna pomaga agentowi poprawić swoją wydajność i dostosować się do ludzkich wartości lub preferencji. W szczególności zwiększa pomocność i prawdomówność w generowaniu, jednocześnie minimalizując szkody i stronniczość. Modele Instruct firmy OpenAI, obecnie modele domyślne, są przykładami modeli opartych na RLHF i wdrażanych na dużą skalę . Anthropic, kolejny startup z branży AI założony przez byłych pracowników OpenAI, dąży do zbudowania systemów LLMS, takich jak Claude, które są niezawodne, interpretowalne i sterowalne. Opublikowali oni swoje podejście do RLHF, w tym powiązane dane dotyczące preferencji człowieka . Metoda RLHF jest szczególnie cenna, gdy zdefiniowanie jednoznacznej funkcji nagrody jest trudne lub gdy kluczowe są ludzkie preferencje, subiektywne oceny i niuanse. Jest wykorzystywana w zadaniach przetwarzania języka naturalnego, takich jak konwersacja i generowanie treści, gdzie liczy się subiektywna jakość. RLHF wspomaga moderację treści w mediach społecznościowych poprzez zrozumienie niuansów kontekstowych. Spersonalizowane systemy rekomendacji pomagają dostosować sugestie do indywidualnych upodobań. Jest cenna dla społecznie akceptowalnej i komfortowej robotyki oraz zachowań interakcji człowiek-komputer. Etyczne podejmowanie decyzji pomaga sztucznej inteligencji w dostosowaniu się do ludzkich wartości. RLHF poprawia wydajność sztucznej inteligencji w złożonych grach, kreatywnych przedsięwzięciach, takich jak sztuka i muzyka, oraz w opiece zdrowotnej, umożliwiając podejmowanie spersonalizowanych decyzji medycznych. Te zastosowania podkreślają rolę RLHF w dostosowaniu sztucznej inteligencji do złożonej i subiektywnej natury ludzkich preferencji i ocen. RLHF może poprawić wydajność, spójność i różnorodność programów LLM w różnych zadaniach, takich jak generowanie tekstu, streszczanie czy dialog. Modele Instruct to podstawowe modele LLM, które zostały dostrojone przy użyciu podejścia RLHF i jak pokazano na rysunku 15, znacznie przewyższają podstawowe modele LLM.



UWAGA: Modele bazowe to oryginalne, wstępnie wytrenowane modele, które nie zostały dopasowane do konkretnych wartości i generalnie nie nadają się do użytku produkcyjnego.

Struktura RLHF uczy model wykonywania zadań w sposób, w jaki chcieliby tego ludzie, wykorzystując ludzkie informacje zwrotne jako wskazówkę. Ta metoda jest szczególnie istotna w dziedzinach, w których pożądany wynik jest subiektywny lub silnie zależny od kontekstu, takich jak przetwarzanie języka naturalnego (NLP), generowanie treści i systemy podejmowania decyzji. Kluczowe fazy składające się na RLHF przedstawiono na rysunku 16.



Komponenty te działają w tandemie, tworząc solidny system uczenia się, w którym model może uczyć się złożonych zadań zorientowanych na człowieka, wykraczających poza możliwości tradycyjnych podejść uczenia maszynowego. Integracja informacji zwrotnej od człowieka jest kluczowa dla zniwelowania luki między algorytmicznym podejmowaniem decyzji a ludzkim osądem. Przyjrzyjmy się bliżej fazom RLHF:

* Dostrajanie nadzorowane — Ta faza obejmuje trenowanie modelu na zbiorze danych przykładów generowanych przez człowieka. Przykłady te demonstrują pożądane rezultaty lub zachowania, stanowiąc punkt odniesienia dla modelu, z którego może się uczyć. Pomaga to modelowi zrozumieć kontekst i niuanse zadań interpretowane przez człowieka.

* Modelowanie nagród — Na tym etapie trenowany jest oddzielny model, znany jako model nagród, w celu przewidywania jakości wyników generowanych przez model główny. Model nagród jest trenowany z wykorzystaniem ludzkich osądów, często obejmujących oceny lub preferencje dotyczące różnych wyników. Model ten skutecznie przekłada subiektywne oceny ludzkie na ilościowe informacje zwrotne, które model główny może wykorzystać do nauki.

* Optymalizacja polityki proksymalnej (PPO) — PPO to algorytm RL, który iteracyjnie ulepsza politykę modelu głównego (proces decyzyjny). Algorytm aktualizuje politykę modelu, aby zmaksymalizować nagrody przewidywane przez model nagród. PPO został wybrany ze względu na stabilność i wydajność w obsłudze dużych i złożonych modeli.

* Pętla sprzężenia zwrotnego — ta pętla obejmuje ciągły wkład ze strony ewaluatorów, którzy oceniają jakość wyników modelu. Informacja zwrotna jest wykorzystywana do dalszego trenowania modelu nagród, tworząc dynamiczne środowisko uczenia się, w którym model dostosowuje się do zmieniających się ludzkich preferencji i standardów. Pętla zapewnia, że model pozostaje zgodny z ludzkimi oczekiwaniami i może dostosowywać się do zmian.

UWAGA: PPO-ptx to adaptacja algorytmu PPO dostosowana do precyzyjnego dostrajania RLHF. Integruje ona odniesienie do oryginalnego modelu LLM, aby utrzymać wydajność, jednocześnie dopasowując wyniki modelu do ludzkich preferencji. Takie podejście pomaga złagodzić podatek od dopasowania, zapewniając, że LLM pozostaje efektywny i zróżnicowany pod względem wyników po szkoleniu. Zasadniczo PPO-ptx równoważy wiedzę modelu zdobytą przed szkoleniem z nową informacją zwrotną, aby stworzyć wysoce wydajny model LLM, zgodny z ludzkimi wartościami. RLHF może wydawać się pod wieloma względami idealnym rozwiązaniem, ale przedsiębiorstwa muszą być świadome pewnych wyzwań. Przyjrzyjmy się im.

Wyzwania związane z RLHF

RLHF to potężna technika uczenia modeli złożonych zadań, ale wiąże się z wieloma praktycznymi wyzwaniami i ograniczeniami dla przedsiębiorstw. System RLHF wymaga dużej ilości danych o ludzkich preferencjach, które są trudne do uzyskania, ponieważ angażują inne osoby, które nie biorą udziału w procesie szkolenia. Skuteczność RLHF zależy od jakości adnotacji ludzkich, które ludzie potrafią napisać, na przykład gdy dostosowują początkowy model LLM w InstructGPT lub oceniają, jak bardzo podobają im się różne wyniki modelu. Niektóre z tych wyzwań to:

* Złożoność techniczna — wdrożenie RLHF wymaga zaawansowanych umiejętności i wiedzy z zakresu uczenia maszynowego, przetwarzania języka naturalnego i przetwarzania języka naturalnego. Wiąże się to również ze złożonymi procesami konfiguracji i konserwacji, takimi jak konfiguracja architektury modelu, systemów nagród i mechanizmów sprzężenia zwrotnego.

* Intensywność obliczeniowa — modele RLHF wymagają wielu zasobów obliczeniowych, takich jak procesory graficzne i serwery, co może być kosztowne. Zależą one również od jakości i ilości informacji zwrotnych od ludzi, które mogą być trudne do uzyskania i przetworzenia. Z praktycznego punktu widzenia, wiele informacji zwrotnych od ludzi pochodzi od pracowników kontraktowych (lub dorywczych) na platformach crowdsourcingowych, gdzie znalezienie odpowiednio wykwalifikowanych osób w określonych dziedzinach może być trudne. Ponadto zapewnienie zróżnicowanego i obiektywnego zbioru danych do szkolenia może być trudne i wymagać dużych nakładów obliczeniowych.

* Brak skalowalności — modele RLHF są trudne do skalowania w aplikacjach na dużą skalę, wymagając ciągłego sprzężenia zwrotnego od człowieka i rosnących zasobów obliczeniowych. Trudno je również dostosować do różnych domen lub zmieniających się środowisk danych, co skutkuje ograniczoną adaptowalnością i możliwością personalizacji.

* Jakość — modele RLHF są podatne na stronniczość, ponieważ odzwierciedlają subiektywne opinie i potencjalne uprzedzenia osób dostarczających informacje zwrotne. Zapewnienie etycznego użytkowania i obiektywnych wyników jest priorytetem. Utrzymanie spójnej jakości informacji zwrotnych od człowieka może być trudne, ponieważ ludzka ocena może się zmieniać i wpływać na niezawodność i wydajność modelu. Próbując zbudować pomocny model, który unika szkód, istnieje nieodłączne napięcie między tymi dwoma wymiarami. Udzielanie zbyt wielu uprzejmych odpowiedzi, takich jak „Przepraszam, jestem modelem AI i nie mogę panu w tym pomóc" lub podobnych, ogranicza użyteczność modelu. Organizacje muszą zrównoważyć i złagodzić ten problem, korzystając z dodatkowych wskazówek, szkoleń i innych technik uczenia maszynowego, aby tworzyć syntetyczne dane, gdzie to możliwe.

* Koszt — modele RLHF są kosztowne we wdrożeniu i obsłudze. Koszty obejmują infrastrukturę, zasoby obliczeniowe, akwizycję danych i zatrudnienie wykwalifikowanych specjalistów. Istnieją również stałe koszty operacyjne związane z zarządzaniem danymi, aktualizacjami modeli i ciągłą integracją informacji zwrotnych. Koszty te mogą być znaczne, szczególnie w przypadku wdrożeń na dużą skalę.

* Dane — trudno jest stworzyć dobry tekst napisany przez człowieka, odpowiadający na konkretne pytania, ponieważ zazwyczaj oznacza to płacenie pracownikom w niepełnym wymiarze godzin (zamiast użytkownikom produktu lub crowdsourcingowi). Na szczęście ilość danych potrzebna do trenowania modelu nagród dla większości zastosowań RLHF (~50 tys. etykiet preferencji) nie jest aż tak kosztowna. Jest to jednak nadal więcej, niż zazwyczaj mogą sobie pozwolić laboratoria akademickie. Istnieje tylko jeden duży zbiór danych dla RLHF w ogólnym modelu językowym (z Anthropic) i kilka mniejszych zbiorów danych dla konkretnych zadań (takich jak dane podsumowujące z OpenAI). Innym problemem związanym z danymi dla RLHF jest to, że annotatorzy mogą się znacznie różnić, co sprawia, że dane treningowe są bardzo zaszumione bez prawdziwej odpowiedzi.

RLHF oferuje zaawansowane możliwości w nauczaniu modeli do wykonywania złożonych zadań; jednak jego wdrożenie w przedsiębiorstwach jest utrudnione ze względu na złożoność techniczną, zapotrzebowanie na zasoby, problemy ze skalowalnością, względy etyczne i wysokie koszty. Z jednej strony bariery te utrudniają wielu organizacjom wdrażanie i utrzymywanie systemów RLHF w praktyce. Z drugiej strony, ci, którzy potrafią to wdrożyć, zwłaszcza niektóre firmy techniczne, takie jak OpenAI i Anthropic, mogą na tym skorzystać. Zobaczmy, jak można skalować wdrożenie RLHF.

Skalowanie wdrożenia RLHF

Skalowanie wdrożenia RLHF dla LLM wymaga wielopłaszczyznowego podejścia, które równoważy wydajność, różnorodność i kontrolę jakości. Po pierwsze, automatyzacja gromadzenia danych i

wdrożenie efektywnych mechanizmów sprzężenia zwrotnego mają kluczowe znaczenie dla obsługi dużych ilości danych i informacji zwrotnych. Zautomatyzowane systemy mogą gromadzić dane z różnych źródeł lub za pośrednictwem interfejsów zaprojektowanych z myślą o efektywnej interakcji międzyludzkiej. Wykorzystanie dużej, zróżnicowanej grupy ewaluatorów jest niezbędne do uchwycenia szerokiego zakresu perspektyw, co pomaga modelowi być bardziej stabilnym i mniej stronniczym. Aby zapewnić merytoryczną wartość informacji zwrotnej, można zastosować inteligentne strategie doboru próby, takie jak aktywne uczenie się, w celu identyfikacji i priorytetyzacji najbardziej wartościowych instancji do oceny. Paralelizacja i dystrybucja zadań pomiędzy wielu ewaluatorów może znacznie przyspieszyć proces ewaluacji. System może obsługiwać przetwarzanie danych na dużą skalę i trenowanie modeli dzięki skalowalnej infrastrukturze. Wdrożenie środków kontroli jakości, takich jak walidacja krzyżowa między ewaluatorami i algorytmami, w celu wykrywania błędów i utrzymania jakości oraz spójności informacji zwrotnej. Regularne monitorowanie i ocena działania modelu może pomóc w zrozumieniu skutków RLHF i ukierunkowaniu ciągłego doskonalenia. Wreszcie, kluczowe są względy etyczne i ograniczanie błędów. Zapewnienie, że informacje zwrotne nie wzmacniają szkodliwych stereotypów, oraz aktywne reagowanie na potencjalne błędy mają kluczowe znaczenie dla tworzenia sprawiedliwych i odpowiedzialnych modeli. Ogólnie rzecz biorąc, skalowanie RLHF dla LLM wymaga kompleksowego podejścia, które integruje strategie techniczne, logistyczne i etyczne, dążąc do stworzenia systemu, który skutecznie uwzględnia opinie ludzi w procesie uczenia się modelu.

Podsumowanie

* Adaptacja modelu powinna być oparta na zestawie przypadków użycia i powinna być ostatecznością dla przedsiębiorstw próbujących ulepszyć model w tych zadaniach.

* Szybkie prace inżynieryjne i RAG muszą współdziałać z dostrajaniem w sposób złożony.

* Prawidłowo przeprowadzone dostrajanie ma duży potencjał wzrostu wydajności i potencjalnych oszczędności kosztów.

* Dostrajanie wiąże się z wysokimi kosztami i należy być świadomym wyzwań, takich jak zapotrzebowanie na dane specyficzne dla danego zadania, zasoby obliczeniowe, ocena wydajności i kwestie etyczne.

* Dostrajanie powinno być przeprowadzane w połączeniu z ewaluacją i często wymaga wielu iteracji, aby uzyskać model gotowy do wdrożenia produkcyjnego.

* Wybór metryk do oceny dostrojonych modeli w dużej mierze zależy od konkretnego zastosowania i celów modelu.

* Głównymi technikami adaptacji modelu, które są bardziej opłacalne, są dostrajanie nadzorowane (SFT), dostrajanie efektywne parametrycznie (PEFT) i adaptacja niskiego rzędu (LoRA).