O'REILLY®

Helion

Projektowanie aplikacji LLM

Holistyczne podejście do dużych modeli językowych

Suhas Pai

Tytuł oryginału: Designing Large Language Model Applications: A Holistic Approach to LLMs

Tłumaczenie: Grzegorz Werner

ISBN: 978-83-289-3111-4

Authorized Polish translation of the English edition of *Designing Large Language Model Applications*
ISBN 9781098150501 © 2025 Suhas Pai

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

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

Printed in Poland.

Spis treści

ROZDZIAŁ 5.

Dostosowywanie modeli językowych do własnych potrzeb

W tym rozdziale będziemy kontynuować przygodę z modelami językowymi — przyjrzymy się różnym LLM-om dostępnym do zastosowań komercyjnych i wyjaśnię, jak wybrać odpowiedni model do konkretnego zadania. Przybliżę także, jak wczytywać LLM-y o różnych rozmiarach i jak przeprowadzać wnioskowanie. Następnie przyjrzymy się różnym strategiom dekodowania podczas generowaniu tekstu. Zbadamy również, jak interpretować wyniki i pośrednie rezultaty modeli językowych z wykorzystaniem narzędzi do interpretacji, takich jak LIT-NLP.

Krajobraz modeli językowych

Nowy LLM pojawia się co kilka dni, a wielu twórców twierdzi, że ich modele są najnowszym krzykiem techniki. Jednak tak naprawdę większość tych modeli nie różni się znacząco od siebie, więc nie trzeba poświęcać zbyt wiele czasu na śledzenie nowych wydań. Informacje o najważniejszych premierach dodaję do repozytorium tej książki na GitHubie (*https://oreil.ly/llm-playbooks*), ale nie gwarantuję, że będzie ono kompletne.

Niemniej warto mieć ogólne rozeznanie w różnych typach dostawców modeli językowych, rodzajach dostępnych LLM-ów oraz implikacjach związanych z prawami autorskimi i licencjonowaniem. Dlatego przyjrzymy się teraz krajobrazowi modeli językowych z tej perspektywy, żeby się zorientować, jakie mamy możliwości.

Kim są dostawcy LLM-ów?

Dostawców LLM-ów można ogólnie podzielić na następujące kategorie:

Firmy oferujące zastrzeżone LLM-y

Należą do nich m.in. takie firmy jak OpenAI (GPT) (*https://oreil.ly/r-lb1*), Google (Gemini) (*https://oreil.ly/KF9Kh*), Anthropic (Claude) (*https://oreil.ly/T5Wvo*), Cohere (*https://oreil.ly/PiKxN*) i AI21 (*https://oreil.ly/Y8T3q*), które trenują własne LLM-y i udostępniają je jako punkt końcowy API („LLM jako usługa"). Wiele z tych firm nawiązało współpracę z dostawcami usług chmurowych, którzy ułatwiają dostęp do ich modeli jako w pełni zarządzanej

usługi. Najważniejsze oferty głównych dostawców chmurowych to Amazon Bedrock (*https://oreil.ly/FVqRj*) i SageMaker JumpStart od Amazona (*https://oreil.ly/e0a59*), Vertex AI od Google'a (*https://oreil.ly/mURoC*) oraz Azure OpenAI od Microsoftu (*https://oreil.ly/Ag1r5*).

Firmy udostępniające LLM-y o otwartym kodzie źródłowym

Do tej grupy należą firmy, które publicznie udostępniają wagi LLM-ów i zarabiają na usługach wdrożeniowych (np. Together AI (*https://oreil.ly/urcAf*)), firmy, których główna działalność skorzystałaby na szerszym wykorzystaniu LLM-ów (np. Cerebras (*https://oreil.ly/2cVYY*)), oraz laboratoria badawcze, które wydają LLM-y od wczesnych dni transformerów (Microsoft, Google, Meta, Salesforce itp.). Warto zauważyć, że niektóre firmy, jak Google, udostępniają zarówno zastrzeżone, jak i otwarte LLM-y.

Samoorganizujące się społeczności open source i organizacje badawcze

Ta kategoria obejmuje pionierską organizację badawczą Eleuther AI (*https://oreil.ly/ZSlbG*) oraz Big Science (*https://oreil.ly/_NlUD*). Organizacje te korzystają z grantów na infrastrukturę obliczeniową.

Środowisko akademickie i instytucje państwowe

Ze względu na wysokie koszty kapitałowe dotychczas niewiele LLM-ów powstało w środowisku akademickim. Przykłady modeli pochodzących od instytucji państwowych lub akademickich to Falcon (*https://oreil.ly/vdhsL*), opracowany przez państwowy Instytut Innowacji Technologicznych w Abu Zabi (*https://oreil.ly/aMwO2*), oraz GLM, stworzony przez Uniwersytet Tsinghua w Pekinie (*https://oreil.ly/K0_zX*).

Tabela 5.1 przedstawia organizacje działające w obszarze LLM-ów, kategorie, do których należą, oraz wydane przez nie wstępnie wytrenowane modele.

Tabela 5.1. Dostawcy LLM-ów

Nazwa	Kategoria	Wydane wstępnie wytrenowane modele
Google	Firma	BERT, MobileBERT, T5, FLAN-T5, ByT5, Canine, UL2, Flan-UL2, Pegasus PaLM, PaLMV2, ELECTRA, Tapas, Switch
Microsoft	Firma	DeBERTa, DialoGPT, BioGPT, MPNet
OpenAI	Firma	GPT-2, GPT-3, GPT-3.5, GPT-4
Amazon	Firma	Titan
Anthropic	Firma	Claude, Claude-2
Cohere	Firma	Cohere Command, Cohere Base
Meta	Firma	RoBERTa, Llama, Llama 2, BART, OPT, Galactica
Salesforce	Firma	CTRL, XGen, EinsteinGPT
MosaicML	Firma (przejęta przez Databricks)	MPT
Cerebras	Firma	Cerebras-GPT, BTLM
Databricks	Firma	Dolly-V1, Dolly-V2
Stability AI	Firma	StableLM
Together AI	Firma	RedPajama

Tabela 5.1. Dostawcy LLM-ów (ciąg dalszy)

Nazwa	Kategoria	Wydane wstępnie wytrenowane modele
Ontocord AI	Organizacja non profit	MDEL
Eleuther AI	Organizacja non profit	Pythia, GPT Neo, GPT-NeoX, GPT-J
Big Science	Organizacja non profit	BLOOM
Uniwersytet Tsinghua	Instytucja akademicka	GLM
Technology Innovation Institute	Instytucja akademicka	Falcon
UC Berkeley	Instytucja akademicka	OpenLLaMA
Adept AI	Firma	Persimmon
Mistral AI	Firma	Mistral
AI21 Labs	Firma	Jurassic
X.AI	Firma	Grok

Odmiany modeli

Każdy model jest zwykle wydawany w kilku wariantach. Powszechną praktyką jest udostępnianie różnych rozmiarów tego samego modelu. Przykładowo Llama 2 występuje w wersjach 7B, 13B i 70B (oznaczenia te odnoszą się do liczby parametrów w modelu).

Obecnie dostawcy LLM-ów rozbudowują swoje wstępnie wytrenowane modele na różne sposoby, aby lepiej dostosować je do zadań użytkowników. Proces ten zazwyczaj obejmuje dostrajanie modelu, często z udziałem nadzoru ludzkiego. Niektóre z tych ćwiczeń dostrajających mogą kosztować miliony dolarów w postaci ludzkich adnotacji. Wstępnie wytrenowane modele, które nie przeszły żadnego procesu rozbudowy, będziemy nazywać modelami bazowymi.

Poniżej opisałem niektóre z popularnych typów rozbudowy modeli.

Modele instrukcyjne

Modele instrukcyjne, czyli dostrajane do wykonywania poleceń, specjalizują się w realizacji instrukcji wyrażonych w języku naturalnym. Choć modele bazowe mają ogromne możliwości, przypominają zbuntowanego nastolatka — skuteczna interakcja z nimi jest możliwa dopiero po żmudnym opracowaniu metodą prób i błędów odpowiednich promptów, które i tak często okazują się nietrwałe. Wynika to stąd, że modele bazowe są trenowane na zadaniach odszumiania lub przewidywania następnego słowa, co różni się od typowych problemów, które chcą rozwiązywać użytkownicy. Dzięki dostrojeniu modelu bazowego do instrukcji wynikowy model może skuteczniej reagować na polecenia człowieka i być bardziej pomocny.

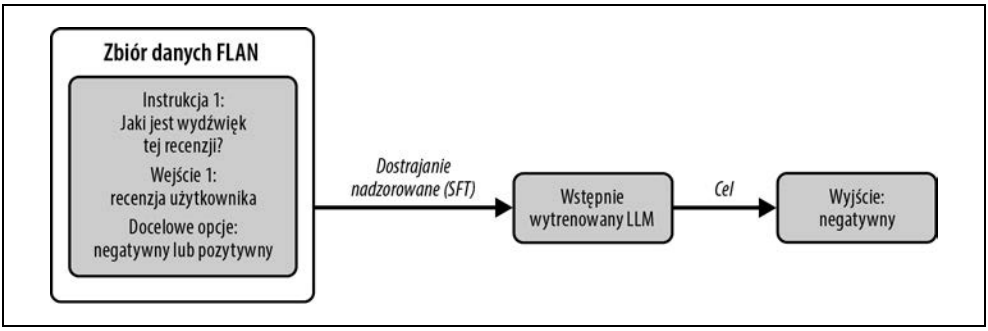
Typowy zbiór danych do dostrajania instrukcyjnego składa się z różnorodnych zadań wyrażonych w języku naturalnym, wraz z parami wejście-wyjście. W rozdziale 6. omówię techniki tworzenia zbiorów danych do dostrajania instrukcyjnego i pokażę, jak przeprowadzić takie dostrajanie na modelu.

Oto przykład z popularnego zbioru danych do treningu instrukcyjnego o nazwie `FLAN` (*https://oreil.ly/YJ_Xr*).

> *Prompt*: „What is the sentiment of the following review? The pizza was ok but the service was terrible. I stopped in for a quick lunch and got the slice special but it ended up taking an hour after waiting several minutes for someone at the front counter and then again for the slices. The place was empty other than myself, yet I couldn't get any help/service. OPTIONS: - negative - positive”
>
> *FLAN*: „Negative”
>
> (Prompt: „Jaki jest wydźwięk tej recenzji? Pizza była w porządku, ale obsługa fatalna. Wpadłem na szybki lunch i zamówiłem zestaw z kawałkiem pizzy, ale ostatecznie zajęło to godzinę, bo musiałem czekać sporo czasu na kogoś przy ladzie, a potem znów na pizzę. Lokal był pusty, byłem tam tylko ja, a mimo to nie mogłem uzyskać żadnej pomocy ani obsługi. OPCJE: - negatywny - pozytywny”
>
> *FLAN*: „Negatywny”)

W tym przykładzie dane wejściowe składają się z instrukcji „What is the sentiment of the following review?” („Jaki jest wydźwięk tej recenzji?”), wyrażonej w sposób naturalny dla człowieka, wraz z danymi wejściowymi i wyjściowymi. Dane wejściowe to właściwa recenzja, a dane wyjściowe to rozwiązanie zadania, wygenerowane przez model lub oznaczone przez człowieka.

Proces dostrajania instrukcyjnego został pokazany na rysunku 5.1.



Rysunek 5.1. Proces dostrajania instrukcyjnego

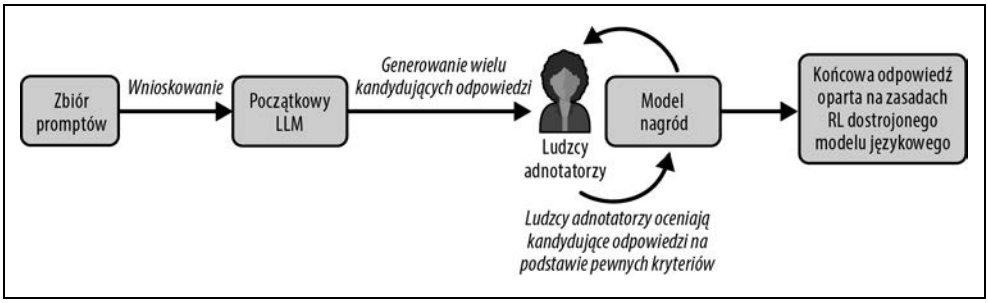
Dostrajanie instrukcyjne to jedna z kilku technik należących do szerszej kategorii dostrajania nadzorowanego (ang. *supervised fine-tuning*, SFT). Metody SFT poprawiają zdolność modelu w zakresie efektywnego realizowania zadań zlecanych przez użytkownika, można też wykorzystać je do zmniejszenia potencjalnej szkodliwości modelu poprzez trening na zbiorach danych dotyczących bezpieczeństwa, które pomagają dostosować wyniki modelu do wartości i preferencji jego twórców.

Bardziej zaawansowane techniki osiągania takiego dostosowania obejmują metody oparte na uczeniu ze wzmacnianiem, takie jak uczenie ze wzmacnianiem z informacją zwrotną od człowieka

(ang. *reinforcement learning from human feedback*, RLHF) oraz uczenie ze wzmacnianiem z informacją zwrotną od sztucznej inteligencji (ang. *reinforcement learning from AI feedback*, RLAIF).

W treningu RLHF ludzcy adnotatorzy wybierają lub oceniają kandydujące odpowiedzi na podstawie określonych kryteriów, takich jak pomocność i nieszkodliwość. Te oceny są wykorzystywane do iteracyjnego trenowania modelu nagród, co ostatecznie prowadzi do większej kontroli nad modelem językowym, na przykład poprzez odmowę odpowiedzi na niewłaściwe prośby użytkowników.

Proces treningu RLHF został pokazany na rysunku 5.2.



Rysunek 5.2. Uczenie przez wzmacnianie z informacją zwrotną od człowieka

Technikę RLHF i inne metody dostrajania omawiam szczegółowo w rozdziale 8.

Zamiast polegać na informacjach zwrotnych od ludzi, w procesie dostrajania można również wykorzystać LLM-y do wybierania wyników według ich zgodności z zestawem zasad (np. unikanie rasizmu, bycie uprzejmym). Tę technikę, nazwaną RLAIF, wprowadziła firma Anthropic. Ludzie dostarczają tu jedynie pożądany zestaw zasad i wartości (określany jako Constitutional AI — *https://oreil.ly/d8FeW*), a model językowy sam ocenia, czy jego wyniki są z nimi zgodne.

Modele dostrojone do wykonywania instrukcji często mają w nazwie przyrostek „instruct", na przykład RedPajama-Instruct.

Dostrajanie modeli może mieć skutki uboczne

Czy zawsze korzystniej jest używać wariantu modelu dostrojonego instrukcjami zamiast modelu bazowego? W większości przypadków tak. Należy jednak pamiętać, że każde dostrajanie modelu bazowego nieuchronnie powoduje pewne regresje, co oznacza utratę niektórych możliwości modelu bazowego.

Przykład tego zjawiska przedstawili Chung i współpracownicy (*https://oreil.ly/-p0R4*). Zauważyli, że dostrajanie instrukcyjne z użyciem zbioru danych `FLAN` pogorszyło zdolność modelu do posługiwania się tzw. łańcuchem myśli (ang. *chain of thought*, CoT), co ma kluczowe znaczenie w zadaniach wymagających rozumowania. Zaobserwowali jednak również, że dodanie danych CoT do zbiorów używanych do dostrajania instrukcyjnego poprawiło zdolności rozumowania modelu w porównaniu z wariantem bazowym.

Efekty uboczne dostrajania instrukcyjnego nie zostały jeszcze dokładnie zbadane, dlatego warto eksperymentować z modelem bazowym i sprawdzać, czy nie tracimy jakichś istotnych funkcji.

Podobnie modele strojone dostosowawczo (ang. *alignment-tuned*) są kalibrowane tak, aby odpowiadały na pytania użytkowników zgodnie z zasadami, wartościami i etyką dostawcy modelu językowego. Wartości te mogą nie być tożsame z wartościami wyznawanymi przez Ciebie lub Twoją organizację.

We wszystkich tych przypadkach możesz przeprowadzić własne dostrajanie instrukcyjne lub dostosowawcze na modelu bazowym. Szczegóły tego procesu omawiam w kolejnych trzech rozdziałach. Przeanalizujemy również, w jakich sytuacjach warto przeprowadzać własne dostrajanie instrukcyjne lub dostosowawcze.

Modele czatowe

Modele czatowe (konwersacyjne) to strojone instrukcyjnie systemy do prowadzenia wieloetapowego dialogu. Przykładami takich modeli są ChatGPT, Llama 2-Chat, MPT-Chat i OpenAssistant.

Modele z długim kontekstem

Jak wyjaśniłem w rozdziale 1., modele językowe oparte na architekturze transformera mają ograniczoną długość kontekstu. Przypomnijmy, że długość kontekstu zazwyczaj odnosi się do sumy liczby tokenów wejściowych i wyjściowych przetwarzanych przez model podczas jednego wywołania. Typowe długości kontekstu współczesnych modeli językowych wahają się od 8000 do 128 000 tokenów, przy czym niektóre warianty modelu Gemini obsługują ponad milion tokenów. Niektóre modele są udostępniane w wariancie z długim kontekstem, na przykład GPT 3.5 ma domyślny rozmiar kontekstu 4K, ale istnieje również wariant z kontekstem 16K. Model MPT (*https://oreil.ly/wKqdL*) również ma wariant z długim kontekstem, który został wytrenowany na długości kontekstu 65K, ale potencjalnie może być używany do jeszcze dłuższych kontekstów podczas wnioskowania.

Modele z długim kontekstem — nie ma nic za darmo

Wykazano (*https://oreil.ly/PSD_k*), że trafność modeli nie utrzymuje się na stałym poziomie wraz ze wzrostem długości kontekstu. LLM-y mają tendencję do zapominania informacji znajdujących się w środkowej części okna kontekstu. Wynika to z charakterystyki dokumentów, na których trenowane są modele — najistotniejszy kontekst potrzebny do przewidzenia kolejnego tokena znajduje się zazwyczaj na początku lub końcu analizowanego fragmentu. W swoich eksperymentach zaobserwowałem, że dla większości modeli punkt krytyczny to kontekst o długości 8000 tokenów — powyżej tej wartości trafność zaczyna spadać. Nie można też po prostu wypełnić całego kontekstu instrukcjami, bo LLM-y potrafią przetworzyć tylko ograniczony zestaw instrukcji w prompcie, po którego przekroczeniu wyniki się pogarszają.

Jednakże modele z długim kontekstem to obszar, w którym obserwujemy najszybsze postępy. Claude i Gemini wykazały istotną poprawę w utrzymywaniu trafności przy długich kontekstach.

Opracowano różne testy do pomiaru trafności modeli w przypadku długich kontekstów, w tym testy „igły w stogu siana" (*https://oreil.ly/aop_Q*). W rozdziale 12. omawiam ograniczenia tych metod ewaluacji i proponuję bardziej kompleksowe podejścia do oceny modeli.

Modele dostosowane do dziedziny lub zadania

Dostawcy LLM-ów często dostrajają swoje modele do konkretnych zadań, takich jak streszczanie tekstów czy analiza odczuć w kontekście finansowym. Tworzą też destylowane wersje modeli, które są dostrajane z wykorzystaniem wyników większego modelu w określonym zadaniu. Przykładami takich specjalistycznych modeli są FinBERT (*https://oreil.ly/uKUAp*), dostrojony do analizy odczuć w tekstach finansowych, oraz UniversalNER (*https://oreil.ly/8A0pn*), który zoptymalizowano do rozpoznawania nazwanych encji.

Otwarte modele językowe

„Otwartoźródłowe" to termin często używany na ogólne określenie modeli, których pewne aspekty są publicznie dostępne. Zdefiniujmy go następująco:

> Oprogramowanie udostępniane na licencji pozwalającej użytkownikom na jego *badanie*, *używanie*, *modyfikowanie* i *redystrybucję* w dowolnym *celu*.

Bardziej formalną i kompleksową definicję oprogramowania otwartoźródłowego można znaleźć w oficjalnej definicji Open Source Initiative (*https://oreil.ly/7cezH*).

Aby model językowy można było uznać za w pełni otwarty, konieczne jest opublikowanie wszystkich poniższych elementów:

Wagi modelu

Obejmuje to wszystkie parametry modelu oraz jego konfigurację. Dostęp do nich umożliwia dodawanie lub modyfikowanie parametrów modelu w dowolny sposób. Zaleca się również udostępnianie punktów kontrolnych modelu z różnych etapów treningu.

Kod modelu

Udostępnienie samych wag modelu przypomina dostarczenie pliku wykonywalnego bez kodu źródłowego. Kod modelu obejmuje nie tylko kod treningowy i ustawienia hiperparametrów, ale także kod używany do przetwarzania danych treningowych. Udostępnienie informacji o konfiguracji infrastruktury znacznie ułatwia odtworzenie modelu. W większości przypadków, nawet przy pełnej dostępności kodu, modele mogą być trudne do odtworzenia ze względu na ograniczenia zasobów i niedeterministyczny charakter treningu.

Dane treningowe

Dane użyte do wytrenowania modelu, a najlepiej także informacje lub kod dotyczący ich pozyskania. Zaleca się również udostępnianie danych z różnych etapów przetwarzania oraz kolejności, w jakiej były one podawane do modelu. Dane treningowe są elementem najrzadziej publikowanym przez twórców modeli. Dlatego większość modeli otwartoźródłowych nie jest w *pełni otwarta*, ponieważ zbiór danych nie jest publiczny.

Dane treningowe często nie są udostępniane z powodów konkurencyjnych. Jak omówiłem w rozdziałach 3. i 4., większość dzisiejszych dużych modeli językowych wykorzystuje warianty tej samej architektury i kodu treningowego. Czynnikiem wyróżniającym może być często zawartość zbiorów danych i ich przetwarzanie wstępne. Część danych treningowych może być pozyskana na podstawie umowy licencyjnej, która zabrania twórcy modelu publicznego udostępniania tych danych.

Innym powodem nieudostępniania danych treningowych są nierozwiązane kwestie prawne, szczególnie dotyczące praw autorskich. Przykładowo zbiór danych `The Pile` stworzony przez Eleuther AI nie jest już dostępny pod oficjalnym linkiem, ponieważ zawiera tekst z książek objętych prawami autorskimi (zbiór Books3). Warto zauważyć, że `The Pile` jest przetworzony, więc książki nie są w formie czytelnej dla człowieka i nie da się ich łatwo odtworzyć, gdyż są podzielone, przemieszane i połączone.

Większość danych treningowych pochodzi z otwartego internetu i może zawierać treści przemocowe lub seksualne, które są zabronione w niektórych częściach świata. Mimo najlepszych intencji i rygorystycznego filtrowania część tych danych może nadal znajdować się w końcowym zbiorze. Z tego powodu wiele zbiorów danych, które wcześniej były otwarte, przestało być dostępnych — przykładem są zbiory obrazów LAION.

Ostatecznie to licencja, na której model został udostępniony, określa warunki jego użytkowania, modyfikowania lub ponownego udostępniania w oryginalnej lub zmodyfikowanej formie.

Ogólnie rzecz biorąc, otwarte modele językowe są dystrybuowane na trzech rodzajach licencji:

Niekomercyjne

Te licencje zezwalają jedynie na badania i użytek osobisty, a zabraniają wykorzystania modelu do celów komercyjnych. W wielu przypadkach dostęp do modelu jest ograniczony poprzez formularz aplikacyjny, w którym użytkownik musi uzasadnić potrzebę dostępu, przedstawiając przekonujący przypadek użycia do celów badawczych.

Copyleft

Ten typ licencji zezwala na użytek komercyjny, ale wymaga, aby cały kod źródłowy lub prace pochodne były udostępniane na tej samej licencji, co utrudnia tworzenie zastrzeżonych modyfikacji. Stopień, w jakim ten warunek ma zastosowanie, zależy od konkretnej używanej licencji.

Permisywne

Ten typ licencji zezwala na użytek komercyjny, w tym modyfikowanie i redystrybucję w zastrzeżonych aplikacjach, czyli nie ma obowiązku, aby redystrybucja miała otwarte źródła. Niektóre licencje w tej kategorii zezwalają również na patentowanie.

Tworzone są nowe rodzaje licencji, które ograniczają wykorzystanie modelu w konkretnych przypadkach, często ze względów bezpieczeństwa. Przykładem jest licencja Open RAIL-M (*https://oreil.ly/2UVMe*), która zabrania wykorzystania modelu w takich przypadkach jak udzielanie porad medycznych, wymiar sprawiedliwości, procedury imigracyjne i azylowe itp. Pełną listę ograniczeń można znaleźć w załączniku A do licencji.

Jeśli zamierzasz wykorzystywać otwarte LLM-y w swojej organizacji do celów komercyjnych, najlepiej będzie skorzystać z modeli o licencjach permisywnych. Popularne przykłady licencji permisywnych to Apache 2.0 i licencja MIT.

Popularną klasą licencji używanych do dystrybucji otwartych LLM-ów jest Creative Commons (CC) (*https://oreil.ly/PQy6D*). Licencje te mają nazwy takie jak CC-BY-NC-SA itp. Oto prosty sposób na zapamiętanie, co oznaczają te nazwy:

BY

Jeśli licencja zawiera to oznaczenie, wymagane jest podanie autora. Jeśli zawiera CC-BY, oznacza to, że licencja jest permisywna.

SA

Jeśli licencja zawiera ten warunek, oznacza to, że redystrybucja powinna odbywać się na zasadach takich samych jak określone w tej licencji. Innymi słowy, jest to licencja typu copyleft.

NC

NC oznacza *noncommercial* (niekomercyjny). Jeśli więc licencja zawiera ten warunek, model może być używany tylko do celów badawczych lub osobistych.

ND

ND oznacza *no derivatives* (zakaz tworzenia dzieł pochodnych). Jeśli licencja zawiera ten warunek, nie wolno rozpowszechniać zmodyfikowanych wersji modelu.

Obecnie za modele otwartoźródłowe uważa się te, które mają otwarte wagi i kod źródłowy oraz są udostępniane na licencji pozwalającej na redystrybucję między dowolnymi odbiorcami i w dowolnym celu. Można jednak argumentować, że dostęp do danych treningowych jest również kluczowy dla analizy i badania modelu, co jest częścią definicji otwartoźródłowości, którą przedstawiłem wcześniej.

W tabeli 5.2 przedstawiłem różne LLM-y, licencje, na jakich są wydawane, oraz ich dostępne rozmiary i warianty. Warto zauważyć, że LLM może być dostrojony instrukcyjnie lub konwersacyjnie przez inny podmiot niż ten, który przeprowadził wstępne trenowanie modelu.

Jak wybrać LLM odpowiedni do określonego zadania?

Biorąc pod uwagę mnogość dostępnych opcji, jak wybrać LLM najodpowiedniejszy do danego zadania? W zależności od sytuacji należy wziąć pod uwagę wiele kryteriów, w tym następujące:

Koszty

Obejmują one koszty wnioskowania lub dostrajania, a także koszty związane z tworzeniem infrastruktury programowej, monitorowaniem i obserwowalnością, wdrażaniem i utrzymaniem (określane łącznie jako LLMOps).

Czas na token wyjściowy (TPOT) (*https://oreil.ly/mEDRt)*

Jest to miara szybkości generowania tekstu z perspektywy użytkownika końcowego.

Tabela 5.2. Lista dostępnych LLM-ów

Nazwa	Dostępność	Rozmiary	Warianty
GPT-4	Zastrzeżony	Nieznane	GPT-4 z kontekstem 32K, GPT-4 z kontekstem 8K
GPT-3.5 Turbo	Zastrzeżony	Nieznane	GPT-3.5 z kontekstem 4K, GPT-3.5 z kontekstem 16K
Claude Instant	Zastrzeżony	Nieznane	-
Claude 2	Zastrzeżony	Nieznane	-
MPT	Apache 2.0	1 mld, 7 mld, 30 mld	MPT 65K storywriter
CerebrasGPT	Apache 2.0	111 mln, 256 mln, 590 mln, 1,3 mld, 2,7 mld, 6,7 mld, 13 mld	CerebrasGPT
Stability LM	CC-BY-SA	7 mld	-
RedPajama	Apache 2.0	3 mld, 7 mld	RedPajama-INCITE-Instruct, RedPajama-INCITE-Chat
GPT-Neo X	Apache 2.0	20 mld	-
BLOOM	Otwarty, ograniczone użycie	176 mld	BLOOMZ
Llama	Otwarty, nie do użytku komercyjnego	7 mld, 13 mld, 33 mld, 65 mld	-
Llama 2	Otwarty, do użytku komercyjnego	7 mld, 13 mld, 70 mld	Llama 2-Chat
Zephyr	Apache 2.0	7 mld	-
Gemma	Otwarty, ograniczone użycie	2 mld, 7 mld	Gemma-Instruction Tuned

Skuteczność zadaniowa

Odnosi się do wymagań w zakresie realizacji zadań i odpowiednich miar, takich jak precyzja czy dokładność. Jaki poziom skuteczności jest wystarczający?

Rodzaje zadań

Charakter zadań, do których będzie używany LLM, np. streszczanie, odpowiadanie na pytania, klasyfikowanie.

Wymagane możliwości

Przykłady obejmują rozumowanie arytmetyczne, rozumowanie logiczne, planowanie, dekompozycję zadań itp. Wiele z tych możliwości, o ile rzeczywiście istnieją, to *emergentne właściwości* LLM-ów, o czym mówiłem w rozdziale 1., które nie występują w mniejszych modelach.

Licencjonowanie

Można korzystać tylko z tych modeli, które pozwalają na dany sposób użytkowania. Nawet modele, które wyraźnie zezwalają na użytek komercyjny, mogą mieć ograniczenia dotyczące niektórych rodzajów zastosowań. Na przykład, jak już wspomniałem, licencja Big Science OpenRAIL-M ogranicza wykorzystanie modelu w przypadkach związanych z wymiarem sprawiedliwości, imigracją lub procedurami azylowymi.

Wewnętrzne talenty ML/MLOps

Kompetencje wewnętrznego zespołu określają, na jakie dostosowania możemy sobie pozwolić. Na przykład, czy mamy wystarczająco dużo wykwalifikowanych pracowników, żeby zbudować system optymalizacji wnioskowania?

Inne kryteria niefunkcjonalne

Obejmują takie aspekty jak bezpieczeństwo, ochrona danych oraz prywatność. Dostawcy usług chmurowych i startupy już wdrażają rozwiązania, które mogą sprostać tym wyzwaniom.

Ćwiczenie

Przygotuj uporządkowaną listę priorytetów dla swojej aplikacji i określ, które z nich są stałe, a które elastyczne. Na przykład precyzja musi wynosić co najmniej X, a czas przetwarzania nie może przekraczać Y.

Na podstawie ustalonych priorytetów jaki model LLM byś wybrał?

Konieczne może być dokonanie wyboru między modelami zastrzeżonymi a otwartymi.

LLM-y otwarte a zastrzeżone

Dyskusje na temat zalet oprogramowania otwartoźródłowego w porównaniu z zastrzeżonym trwają w branży technologicznej od kilkudziesięciu lat. Obecnie stają się one coraz bardziej istotne również w kontekście LLM-ów. Głównymi zaletami modeli otwartoźródłowych są przejrzystość i elastyczność, a niekoniecznie niższy koszt. Samodzielne hostowanie takich modeli może wiązać się z dużymi nakładami inżynieryjnymi oraz kosztami obliczeniowymi i pamięciowymi, a korzystanie z usług zarządzanych nie zawsze dorównuje modelom zamkniętym pod względem opóźnień, przepustowości i kosztów wnioskowania. Ponadto wiele modeli otwartoźródłowych nie jest łatwo dostępnych w ramach usług zarządzanych i innych opcji wdrażania przez strony trzecie. Sytuacja ta z pewnością ulegnie zmianie, ale tymczasem warto dokładnie przeanalizować koszty korzystania z każdego (typu) modelu w konkretnym przypadku.

Elastyczność modeli otwartoźródłowych ułatwia debugowanie, interpretację i rozszerzanie LLM-a poprzez dowolne metody treningu i dostrajania, zamiast ograniczonych opcji udostępnianych przez dostawcę. Pozwala to lepiej dostosować model do własnych preferencji i wartości, zamiast narzuconych przez dostawcę. Pełny dostęp do prawdopodobieństw tokenów (logitów) daje ogromne możliwości, co pokaże w dalszej części książki.

Dostępność modeli otwartoźródłowych umożliwiła zespołom tworzenie modeli i aplikacji, które mogą nie być opłacalne dla większych firm nastawionych na zysk. Przykładem jest dostosowywanie modeli do obsługi języków o ograniczonych zasobach piśmienniczych (takich, które nie mają znaczącej reprezentacji w internecie, jak regionalne języki Indii czy języki rdzennych mieszkańców Kanady). Dobrym przykładem jest model Kannada Llama (*https://oreil.ly/hoBQ1*), zbudowany na bazie Llama 2 poprzez ciągłe wstępne trenowanie i dostrajanie na tokenach z kannady, regionalnego języka Indii.

Nie wszystkie modele otwartoźródłowe są w pełni transparentne. Jak już wspomniałem, większość firm komercyjnych udostępniających takie modele nie ujawnia szczegółów dotyczących zbiorów treningowych. Na przykład Meta nie ujawniła wszystkich informacji o danych użytych do treningu modelu Llama 2. Znajomość tych zbiorów pomaga ocenić ryzyko zanieczyszczenia zbioru testowego i zrozumieć, jakiej wiedzy można oczekiwać od modelu.

Kiedy pisałem tę książkę, modele otwartoźródłowe, takie jak Llama 3.2 i DeepSeek v3, w dużej mierze dorównywały najnowocześniejszym modelom zastrzeżonym od OpenAI czy Anthropic. Pojawia się jednak nowa luka między modelami zamkniętymi a otwartoźródłowymi w dziedzinie modeli rozumowania, takich jak o3 od OpenAI, które wykorzystują techniki obliczeniowe w czasie wnioskowania (co omawiam w rozdziale 8.). W tej książce przedstawię scenariusze, w których modele otwartoźródłowe mają przewagę.

Zawsze warto sprawdzić, czy dostawca modelu ma aktywną społeczność programistów na GitHubie, Discordzie lub Slacku oraz czy zespół rozwojowy udziela się na tych kanałach, odpowiadając na komentarze i pytania użytkowników. Zalecam preferowanie modeli z aktywnymi społecznościami, o ile spełniają one podstawowe kryteria.

Ocena LLM-ów

Zacznijmy od ostrzeżenia: ocena LLM-ów jest obecnie chyba najtrudniejszym zadaniem w dziedzinie modeli językowych. Obecne metody testów porównawczych są niedoskonałe, łatwe do obejścia i trudne w interpretacji. Niemniej jednak testy porównawcze wciąż stanowią użyteczny punkt wyjścia w procesie ewaluacji. Przyjrzymy się najpierw dostępnym publicznie testom, a następnie zobaczymy, jak można tworzyć bardziej kompleksowe wewnętrzne systemy oceny.

Do oceny wydajności LLM-ów w konkretnych zadaniach używa się wielu różnych zbiorów danych testowych, które sprawdzają rozmaite umiejętności. Nie wszystkie będą istotne w danej sytuacji, więc możesz skupić się na tych testach, które sprawdzają umiejętności kluczowe dla Twojego projektu.

Rankingi w tych testach porównawczych zmieniają się bardzo często, szczególnie jeśli oceniane są tylko modele otwartoźródłowe. Nie oznacza to jednak, że musisz zmieniać używany model za każdym razem, gdy pojawia się nowy lider. Zwykle różnice między najlepszymi modelami są marginalne. Wybór konkretnego LLM-a zazwyczaj nie jest najważniejszym czynnikiem decydującym o sukcesie Twojego zadania. Lepiej poświęcić czas na oczyszczenie i zrozumienie danych, co nadal stanowi najistotniejszy element projektu.

Przyjrzyjmy się teraz kilku popularnym metodom oceny LLM-ów stosowanym w branży.

Eleuther AI LM Evaluation Harness

LM Evaluation Harness (*https://oreil.ly/SiOXq*) pozwala na testy porównawcze w ponad 400 różnych zadaniach, oceniając rozmaite umiejętności, m.in. odpowiadanie na pytania otwarte, rozumowanie arytmetyczne i logiczne, zadania językowe, tłumaczenie maszynowe i wykrywanie

toksycznego języka. Możesz użyć tego narzędzia do oceny dowolnego modelu dostępnego w repozytorium Hugging Face Hub (*https://oreil.ly/IHd22*), platformie zawierającej tysiące wstępnie wytrenowanych i dostrojonych modeli.

Oto przykład z jednego z zadań testowych o nazwie `bigbench_formal_fallacies_syllogisms_negation`:

```
{
   "input": "\"Some football fans admire various clubs, others love
   only a single team. But who is a fan of whom precisely? The
   following argument pertains to this question: First premise: Mario
   is a friend of FK \u017dalgiris Vilnius. Second premise: Being a
   follower of F.C. Copenhagen is necessary for being a friend of FK
   \u017dalgiris Vilnius. It follows that Mario is a follower of F.C.
   Copenhagen.\"\n Is the argument, given the explicitly stated
   premises, deductively valid or invalid?",
   "target_scores": {
      "valid": 1,
      "invalid": 0
}
```

W tym zadaniu model ma za zadanie wykryć błędy logiczne poprzez sprawdzenie poprawności argumentu na podstawie określonych przesłanek.

Ćwiczenie

Przetestujmy kilka modeli na tym zadaniu. Postępuj zgodnie z instrukcjami (*https://oreil.ly/mZdGA*), aby zainstalować platformę testową. Następnie możesz uruchomić poniższy kod, aby ocenić model Falcon 7B:

```
lm_eval --model hf-causal \
        --model_args pretrained=tiiuae/falcon-7b \
        --tasks bigbench_formal_fallacies_syllogisms_negation \
        --device cuda:0
```

Wypróbuj to dla kilku innych modeli 7B, w tym Llama, Gemma, Mistral, MPT i RedPajama, zarówno w wersjach podstawowych, jak i dostrojonych instrukcyjnie, jeśli są dostępne.

Czy zauważasz duże różnice w trafności między tymi modelami?

Dodatkowo przygotuj dziesięć własnych pytań do tego samego zadania (możesz użyć modelu do wygenerowania propozycji pytań, które następnie zmodyfikujesz) dotyczących różnych dziedzin. Czy dla Twoich pytań modele wykazują taki sam poziom trafności jak w przypadku testów wzorcowych?

Narzędzie to można również wykorzystać do oceny modeli zastrzeżonych. Na przykład modele OpenAI można ocenić w następujący sposób:

```
export OPENAI_API_SECRET_KEY=<Key>
python main.py \
lm_eval --model openai-completions \
        --model_args model=gpt-3.5-turbo \
         --tasks bigbench_formal_fallacies_syllogisms_negation
```

Ćwiczenie

Porównaj modele GPT 4o, 4o-mini, o1 i o3 w zadaniu wykrywania błędów logicznych, uwzględniając zarówno standardowe zestawy testowe, jak i testy przygotowane przez Ciebie. Jak wypadają one względem siebie i jak radzą sobie w porównaniu z modelami otwartoźródłowymi?

Przy wyborze lub opracowywaniu zadania do oceny wydajności zalecam skupienie się na testach sprawdzających umiejętności potrzebne do rozwiązania interesującego Cię problemu, zamiast na samym zadaniu. Na przykład jeśli tworzysz aplikację do streszczania, która wymaga dużo wnioskowania logicznego, lepiej skoncentrować się na testach bezpośrednio sprawdzających zdolności logicznego rozumowania niż na tych, które oceniają jakość streszczeń.

Ranking otwartych modeli językowych Hugging Face

Kiedy pisałem tę książkę, ranking Open LLM Leaderboard (*https://oreil.ly/tspBY*) wykorzystywał narzędzie LM Evaluation Harness opracowane przez Eleuther AI do oceny wydajności modeli w sześciu zadaniach wzorcowych:

Massive Multitask Language Understanding (MMLU)

Ten test ocenia LLM w zadaniach wymagających szerokiej wiedzy, czerpiąc z dziedzin takich jak historia Stanów Zjednoczonych, biologia, matematyka i ponad 50 innych przedmiotów w formie pytań wielokrotnego wyboru.

AI2 Reasoning Challenge (ARC)

To zadanie polega na odpowiadaniu na pytania wielokrotnego wyboru z zakresu nauk przyrodniczych na poziomie szkoły podstawowej wymagające złożonego rozumowania i wiedzy ogólnej.

Hellaswag

Ten test sprawdza zdolność modelu do wnioskowania zdroworozsądkowego, przedstawiając mu sytuację i prosząc o przewidzenie, co może się wydarzyć dalej, poprzez wybór jednej spośród podanych opcji.

TruthfulQA

To zadanie ocenia zdolność modelu do udzielania odpowiedzi, które nie zawierają fałszywych informacji.

Winogrande

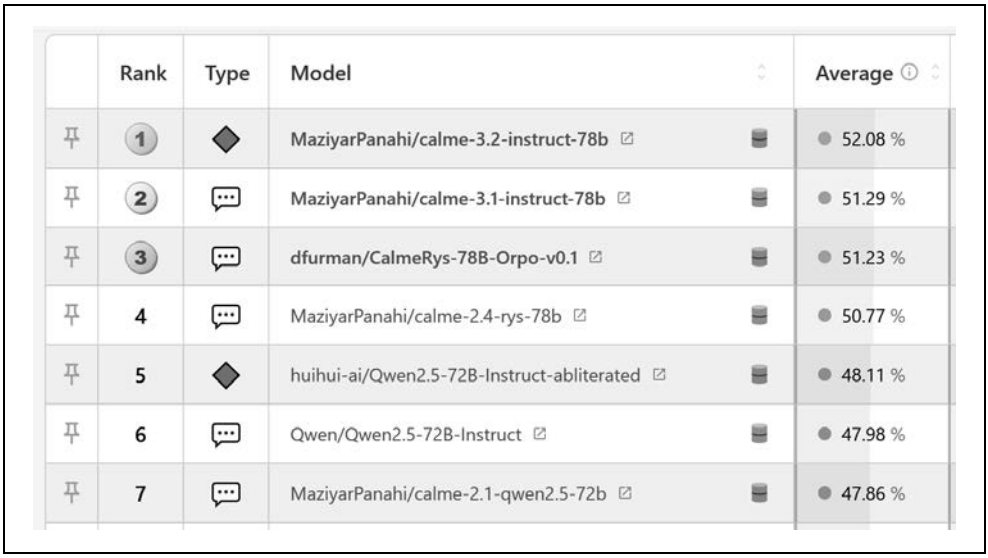
Ten test składa się z pytań z lukami do uzupełnienia, które sprawdzają umiejętność wnioskowania zdroworozsądkowego.

GSM8K

Ten test ocenia zdolność LLM-ów do rozwiązywania zadań matematycznych na poziomie szkoły podstawowej wymagających wykonania sekwencji podstawowych działań arytmetycznych.

Rysunek 5.3 przedstawia zrzut ekranu z rankingiem LLM-ów w momencie pisania tej książki. Oto co możemy zaobserwować:

- Większe modele osiągają lepsze wyniki.
- Warianty modeli dostrojone instrukcyjnie lub poddane specjalistycznemu treningowi radzą sobie lepiej.



	Rank	Type	Model	Average
	1		MaziyarPanahi/calme-3.2-instruct-78b	52.08 %
	2		MaziyarPanahi/calme-3.1-instruct-78b	51.29 %
	3		dfurman/CalmeRys-78B-Orpo-v0.1	51.23 %
	4		MaziyarPanahi/calme-2.4-rys-78b	50.77 %
	5		huihui-ai/Qwen2.5-72B-Instruct-abliterated	48.11 %
	6		Qwen/Qwen2.5-72B-Instruct	47.98 %
	7		MaziyarPanahi/calme-2.1-qwen2.5-72b	47.86 %

Rysunek 5.3. Zrzut ekranu z rankingiem Open LLM Leaderboard

Ćwiczenie

Zachodzi podejrzenie (*https://oreil.ly/fwVEC*), że duża liczba modeli mogła zostać wytrenowana na danych zanieczyszczonych zbiorem GSM8K. Warto zapoznać się z tym zbiorem (*https://oreil.ly/dI87I*), wprowadzić tylko część pytania i sprawdzić, czy modele oceniane w poprzednich ćwiczeniach poprawnie uzupełniają pytanie. Dobrze jest też zmienić liczby w zadaniach i zweryfikować, czy trafność pozostaje taka sama.

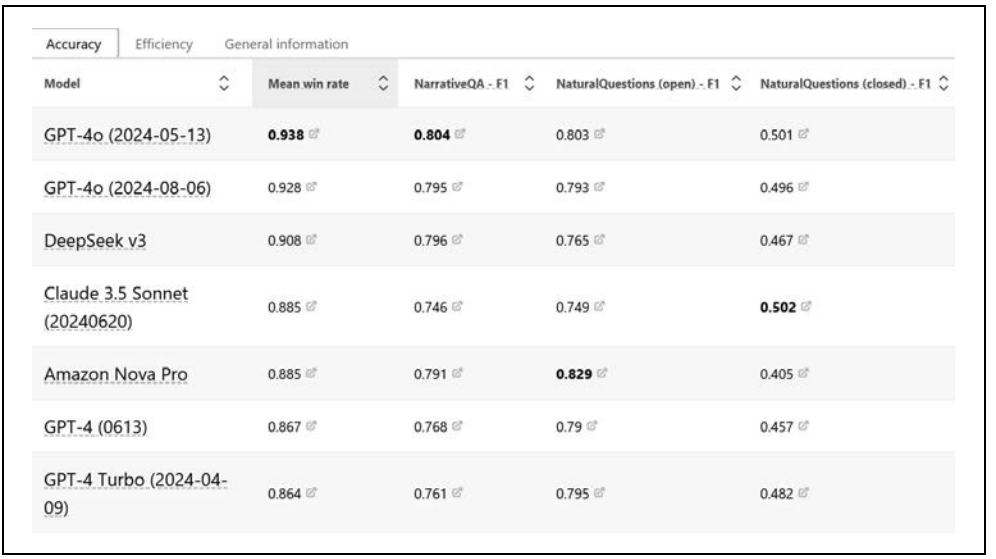
Wiarygodność testów porównawczych stoi pod znakiem zapytania, ponieważ nie ma gwarancji całkowitego oczyszczenia zbioru testowego. Dostawcy modeli optymalizują je również pod kątem rozwiązywania tych testów, co zmniejsza ich wartość jako wiarygodnych wskaźników trafności ogólnej.

HELM

HELM (Holistic Evaluation of Language Models) (*https://oreil.ly/MNHDs*) to opracowany przez Uniwersytet Stanforda kompleksowy model ewaluacyjny, którego celem jest obliczanie szerokiej gamy miar dla różnorodnych zadań wzorcowych. Łącznie obliczanych jest 59 miar testujących dokładność, kalibrację, odporność, sprawiedliwość, stronniczość, toksyczność,

wydajność, jakość streszczeń, naruszenia praw autorskich i wiele innych aspektów. Testowane zadania obejmują odpowiadanie na pytania, tworzenie streszczeń, klasyfikowanie tekstu, wyszukiwanie informacji, analizę odczuć i wykrywanie toksyczności.

Rysunek 5.4 przedstawia zrzut ekranu z rankingiem HELM w czasie pisania tej książki.



Accuracy | Efficiency | General information

Model	Mean win rate	NarrativeQA - F1	NaturalQuestions (open) - F1	NaturalQuestions (closed) - F1
GPT-4o (2024-05-13)	**0.938**	**0.804**	0.803	0.501
GPT-4o (2024-08-06)	0.928	0.795	0.793	0.496
DeepSeek v3	0.908	0.796	0.765	0.467
Claude 3.5 Sonnet (20240620)	0.885	0.746	0.749	**0.502**
Amazon Nova Pro	0.885	0.791	**0.829**	0.405
GPT-4 (0613)	0.867	0.768	0.79	0.457
GPT-4 Turbo (2024-04-09)	0.864	0.761	0.795	0.482

Rysunek 5.4. Zrzut ekranu z rankingiem HELM

Testy porównawcze są zawodne

To samo zadanie można oceniać na wiele sposobów. Weźmy na przykład zadanie MMLU. Pytania w zadaniu MMLU mają cztery możliwe odpowiedzi: A, B, C, D. Jak ocenić trafność w zadaniu odpowiadania na pytania wielokrotnego wyboru?

- Można wybrać token o najwyższym prawdopodobieństwie wyjściowym spośród czterech opcji (A, B, C, D).
- Można wybrać token o najwyższym prawdopodobieństwie wyjściowym z całego słownika i dopasować go do poprawnej odpowiedzi na pytanie (nie do etykiety jak A, B, C, D, ale faktycznej odpowiedzi).
- Można wygenerować znormalizowaną sumę prawdopodobieństw dla sekwencji tokenów utworzonej przez model, gdzie oczekiwaną sekwencją tokenów jest etykieta z następującym po niej tekstem odpowiedzi, i użyć jej do dopasowania poprawnej odpowiedzi (reprezentowanej przez etykietę i tekst odpowiedzi).

Każda z tych metod obliczeniowych może dawać zupełnie inny wynik i wskazywać innych liderów w rankingu. Post na ten temat pojawił się na blogu Hugging Face (*https://oreil.ly/QrBX4*) po tym, jak niektórzy zauważyli rozbieżności między ich wynikami a ocenami zewnętrznymi.

Ranking Elo

Teraz, gdy znasz już ograniczenia oceny ilościowej, przyjrzyjmy się, jak możemy najefektywniej wykorzystać oceny ludzkie. Obiecującym rozwiązaniem jest system rankingowy Elo (*https://oreil.ly/bTD7I*) stosowany do klasyfikacji szachistów.

Organizacja Large Model Systems (LMSYS Org) (*https://oreil.ly/HGVz2*) zaimplementowała opartą na systemie Elo platformę ewaluacyjną o nazwie Chatbot Arena (*https://oreil.ly/evgQX*). Chatbot Arena gromadzi oceny od użytkowników, zapraszając ich do wyboru pomiędzy dwoma losowo dobranymi i anonimowymi modelami językowymi poprzez równoległą rozmowę z nimi. Ranking jest dostępny w sieci (*https://oreil.ly/Y6zmN*), a czołowe miejsca zajmują modele firm takich jak OpenAI, DeepSeek, Google DeepMind i Anthropic.

Rysunek 5.5 przedstawia zrzut ekranu z rankingiem Chatbot Arena w czasie pisania tej książki.

Rank* (UB)	Rank (StyleCtrl)	Model	Arena Score
1	3	Gemini-2.0-Flash-Thinking-Exp-01-21	1384
1	2	Gemini-2.0-Pro-Exp-02-05	1379
1	1	ChatGPT-4o-latest (2025-01-29)	1377
4	2	DeepSeek-R1	1361
4	7	Gemini-2.0-Flash-001	1355
4	2	o1-2024-12-17	1352
7	5	o1-preview	1335
7	7	Qwen2.5-Max	1332
9	8	DeepSeek-V3	1316
9	9	Gemini-2.0-Flash-Lite-Preview-02-05	1309

Rysunek 5.5. Zrzut ekranu z rankingiem Chatbot Arena

Rankingi Elo też mogą być stronnicze

Rankingi Elo nie są panaceum na problem ewaluacji modeli językowych. Uprzedzenia ludzkie mogą znacząco wpływać na ogólne oceny, nawet jeśli modele są oceniane anonimowo.

Według Wu i współpracowników (*https://oreil.ly/3g_qd*) są to następujące objawy stronniczości:

- Ludzie mają tendencję do preferowania dłuższych tekstów.
- Ludzie często przeoczają subtelne problemy z faktami i spójnością, jeśli styl jest autorytatywny lub przekonujący.
- Ludzie mogą być niezdecydowani i skłonni do przyznawania remisów zamiast wybierania zwycięzcy.

- Kolejność prezentowania odpowiedzi modeli może wpływać na oceny ludzkie. Można temu zaradzić, przedstawiając użytkownikowi odpowiedzi w losowej kolejności.

Wu i współpracownicy proponują system ocen multi-Elo, który prosi ludzi o ocenę modeli w trzech różnych wymiarach: użyteczności, dokładności i jakości języka.

Interpretowanie wyników testów porównawczych

Jak interpretować wyniki ewaluacji przedstawione w artykułach naukowych? Postaraj się metodycznie zadać jak najwięcej pytań i sprawdzić, czy odpowiedzi na nie znajdują się w artykule lub innych materiałach. Jako przykład weźmy wykresy ewaluacji modelu Llama 2-chat przedstawione w pracy na temat Llamy 2 (*https://oreil.ly/BcgXs*). W szczególności przyjrzyj się rysunkom 1 i 3, które pokazują, jak Llama 2-Chat wypada pod względem użyteczności i bezpieczeństwa w porównaniu z innymi modelami konwersacyjnymi. Oto niektóre z pytań, które mogą się nasunąć:

- Jak wygląda zbiór danych użyty do ewaluacji? Czy mamy do niego dostęp?
- Jaki jest poziom trudności zbioru testowego? Może model jest konkurencyjny w stosunku do ChatGPT w prostszych przykładach, ale jak radzi sobie z trudniejszymi?
- Jaką część przykładów w zbiorze testowym można uznać za trudne?
- Jakie scenariusze są uwzględnione w zestawie testowym? W jakim stopniu pokrywają się one ze zbiorami użytymi do dostrojenia modelu?
- Jaką definicję bezpieczeństwa przyjęto?
- Czy ocena może być stronnicza dlatego, że modele są oceniane na podstawie konkretnej definicji bezpieczeństwa, pod kątem której trenowano Llamę 2, podczas gdy inne modele mogą mieć inne definicje bezpieczeństwa?

Taka rygorystyczna analiza wyników pomaga lepiej zrozumieć, co jest oceniane i czy ma to związek z możliwościami modelu, których potrzebujesz do własnych zadań. Aby przeprowadzić bardziej rygorystyczną ewaluację LLM-ów, zdecydowanie warto opracować własne, wewnętrzne testy porównawcze.

Nie ufaj ocenom przeprowadzanym przez GPT-4 lub inne LLM-y. Nie wiemy, jakie kryteria oceny stosują te modele, i nie do końca rozumiemy ich tendencyjność.

Rzetelna ocena LLM-ów jest dodatkowo utrudniona przez wrażliwość na prompty oraz probabilistyczny charakter modeli generatywnych. Na przykład często spotykam się z twierdzeniami, że „GPT-4 nie ma zdolności rozumowania", podczas gdy w ocenie nie zastosowano żadnych technik formułowania promptów. W wielu przypadkach okazuje się, że model potrafi wykonać zadanie, jeśli zastosuje się prompt typu CoT. Chociaż prompty ewaluacyjne nie muszą być intensywnie dopracowywane, stosowanie podstawowych technik, takich jak CoT, powinno być standardową praktyką. Ich pomijanie oznacza, że możliwości modelu są niedoszacowane.

Wczytywanie modeli językowych

Choć można wczytywać LLM-y i używać ich do wnioskowanie z zastosowaniem samych procesorów (CPU), do uzyskania akceptowalnej szybkości generowania tekstu potrzebne są karty graficzne (GPU). Wybór odpowiedniej karty graficznej zależy od kosztów, rozmiaru modelu, tego, czy zamierzasz trenować model, czy tylko przeprowadzać wnioskowanie, a także od wsparcia dla optymalizacji. Tim Dettmers opracował świetny schemat blokowy (*https://oreil.ly/t6iPQ*), który pomoże Ci wybrać kartę graficzną najlepiej odpowiadającą Twoim potrzebom.

Przyjrzyjmy się, ile pamięci RAM karty graficznej potrzeba do wczytania LLM-a o danym rozmiarze. LLM-y można wczytywać w różnych precyzjach:

Float32

32-bitowa reprezentacja zmiennoprzecinkowa, gdzie każdy parametr zajmuje 4 bajty pamięci.

Float16

16-bitowa reprezentacja zmiennoprzecinkowa. Na wykładnik zarezerwowanych jest tylko 5 bitów, a nie 8, jak w przypadku Float32. Oznacza to, że korzystanie z Float16 wiąże się z problemami przepełnienia/niedomiaru dla bardzo dużych i małych liczb.

bfloat16 (BF16)

16-bitowa reprezentacja zmiennoprzecinkowa. Podobnie jak we Float32, na wykładnik zarezerwowanych jest 8 bitów, co eliminuje problemy przepełnienia/niedomiaru występujące w przypadku Float16.

Int8

8-bitowa reprezentacja całkowitoliczbowa. Wnioskowanie w trybie 8-bitowym jest o około 20% wolniejsze niż we Float16.

FP8, FP4

8- i 4-bitowa reprezentacja zmiennoprzecinkowa.

Szczegółowo omawiam te formaty w rozdziale 9. Ogólnie rzecz biorąc, do przeprowadzenia wnioskowania na modelu z 7 miliardami parametrów potrzeba około 7 GB pamięci RAM karty graficznej w trybie 8-bitowym i około 14 GB w trybie BF16. Jeśli zamierzasz dostroić cały model, będziesz potrzebować znacznie więcej pamięci.

Hugging Face Accelerate

Możesz używać modeli do wnioskowania nawet wtedy, gdy nie mieszczą się one w pamięci RAM karty graficznej. Umożliwia to opracowana przez Hugging Face biblioteka `accelerate` (*https://oreil.ly/OYdyf*), która wczytuje części modelu do pamięci RAM procesora, gdy pamięć karty graficznej jest zapełniona, a następnie zapisuje części modelu na dysku, gdy wyczerpie się pamięć RAM procesora. Techniczne informacje o działaniu tej biblioteki można znaleźć w artykule *Accelerate Big Model Inference: How Does it Work?* (*https://oreil.ly/J8duc*). Cały proces jest ukryty przed użytkownikiem, więc w celu wczytania dużego modelu wystarczy uruchomić następujący kod:

```
!pip install transformers accelerate
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
tokenizer = AutoTokenizer.from_pretrained("EleutherAI/gpt-neox-20B")
model = GPTNeoForCausalLM.from_pretrained("EleutherAI/gpt-neox-20B")
input_ids = tokenizer("Language models are", return_tensors="pt")
gen_tokens = model.generate(**input_ids, max_new_tokens =1)
```

Ollama

Istnieje wiele narzędzi umożliwiających lokalne wczytywanie LLM-ów, nawet na laptopie. Jedną z takich bibliotek jest `Ollama`, która działa w systemach Windows, Mac i Linux. Przy użyciu tej biblioteki można wczytać modele o wielkości 13 miliardów parametrów, jeśli komputer ma co najmniej 16 GB dostępnej pamięci RAM. `Ollama` obsługuje wiele modeli otwartych, takich jak Mistral, Llama czy Gemma. Udostępnia też interfejs REST API, który można wykorzystać do przeprowadzania wnioskowania i tworzenia aplikacji opartych na LLM-ach. Ponadto oferuje integracje z terminalem i interfejsem użytkownika, co ułatwia budowanie aplikacji dla użytkowników końcowych.

Zobaczmy, jak można wykorzystać model Gemma 2B firmy Google za pomocą biblioteki `Ollama`. Najpierw ze strony *https://oreil.ly/yly44* pobierz jej wersję odpowiednią dla swojego systemu operacyjnego. Następnie pobierz model Gemma na swój komputer, używając polecenia:

```
ollama pull gemma:2b
```

Możesz również utworzyć plik *Modelfile*, który zawiera informacje konfiguracyjne dla modelu. Obejmuje on prompty systemowe i szablony promptów, parametry dekodowania, takie jak temperatura, oraz historię konwersacji. Pełną listę dostępnych opcji znajdziesz w dokumentacji (*https://oreil.ly/ba-1u*).

Przykładowy plik *Modelfile* wygląda tak:

```
FROM gemma:2b

PARAMETER temperature 0.2

SYSTEM """
You are a provocateur who speaks only in limericks.
"""
```

Po utworzeniu pliku *Modelfile* możesz uruchomić model:

```
ollama create local-gemma -f ./Modelfile
ollama run local-gemma
```

Repozytorium tej książki na GitHubie zawiera przykładową aplikację zbudowaną od początku do końca przy użyciu biblioteki `Ollama` i jednej z jej integracji interfejsu użytkownika. Możesz również eksperymentować z podobnymi narzędziami, takimi jak LM Studio (*https://oreil.ly/uFsiR*) i GPT4All (*https://oreil.ly/XUXhq*).

`Ollama` umożliwia też wczytywanie własnych modeli w formacie GGUF (GPT Generated Unified Format).

API wnioskowania w LLM-ach

Choć można samodzielnie wdrożyć LLM, współczesne wnioskowanie obejmuje wiele optymalizacji, z których sporo jest zastrzeżonych, przez co uzyskanie szybkości wnioskowania zbliżonej do rozwiązań komercyjnych wymaga dużego nakładu pracy. Istnieją usługi wnioskowania, takie jak Together AI (*https://oreil.ly/L3zo0*), które umożliwiają korzystanie z otwartych lub własnych modeli za pośrednictwem bezserwerowych punktów końcowych lub osobnych instancji. Inną opcją jest rozwiązanie TGI (Text Generation Inference) od Hugging Face (*https://oreil.ly/XXFpa*), które niedawno zostało ponownie udostępnione (*https://oreil.ly/BJJlY*) na permisywnej licencji open source.

Strategie dekodowania

Teraz, gdy umiesz już wczytać model, przyjrzyjmy się, jak efektywnie generować tekst. W tym celu w ostatnich latach opracowano kilka strategii **dekodowania**. Omówię je szczegółowo.

Dekodowanie zachłanne

Najprostsza forma dekodowania polega na generowaniu tokena o najwyższym prawdopodobieństwie. Wadą tego podejścia jest to, że prowadzi ono do powtarzalności w wynikach. Oto przykład:

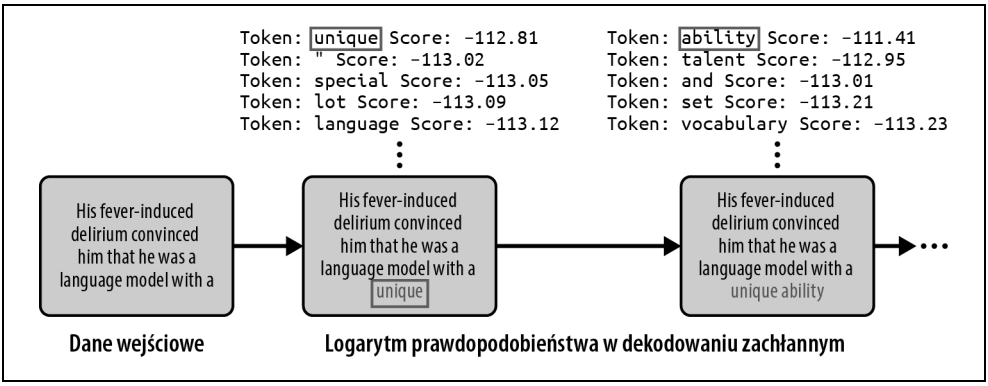
```
input = tokenizer('The keyboard suddenly came to life. It ventured up the',

return_tensors='pt').to(torch_device)
output = model.generate(**inputs, max_new_tokens=50)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

Przekonasz się, że wyniki są powtarzalne. Dlatego dekodowanie zachłanne nie jest dobrym rozwiązaniem, chyba że generujesz bardzo krótkie sekwencje, na przykład pojedynczy token w zadaniu klasyfikacji.

Rysunek 5.6 przedstawia przykład dekodowania zachłannego z użyciem modelu FLAN-T5. Warto zauważyć, że pomijamy kilka świetnych sekwencji, ponieważ jeden z pożądanych tokenów miał nieco niższe prawdopodobieństwo, co sprawiło, że nigdy nie został wybrany.

Wyszukiwanie wiązkowe

Alternatywą dla dekodowania zachłannego jest wyszukiwanie wiązkowe. Ważnym parametrem tego algorytmu jest rozmiar wiązki, *n*. W pierwszym kroku wybiera się *n* tokenów o najwyższym prawdopodobieństwie jako hipotezy. W kolejnych krokach model generuje kontynuacje dla każdej z hipotez. Token wybrany do wygenerowania to ten, którego kontynuacje mają najwyższe skumulowane prawdopodobieństwo.

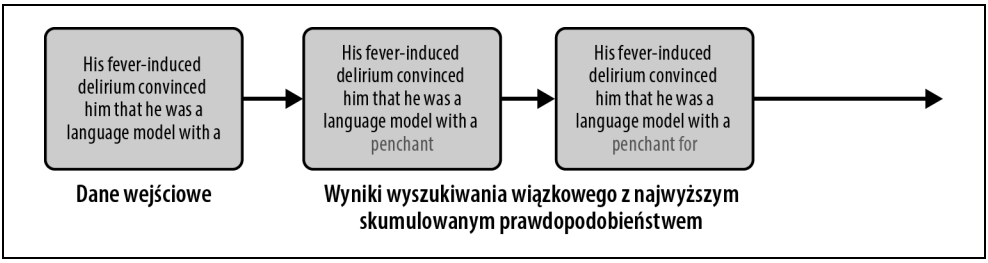


Rysunek 5.6. Dekodowanie zachłanne

W bibliotece `Hugging Face Transformers` rozmiar wiązki określa parametr `num_beams` funkcji `model.generate()`. Oto jak wyglądałby kod dekodowania z użyciem wyszukiwania wiązkowego:

```
output = model.generate(**inputs, max_new_tokens=50, num_beams = 3)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

Na rysunku 5.7 został pokazany przykład wyszukiwania wiązkowego z wykorzystaniem modelu FLAN-T5. Warto zauważyć, że metoda ta nie rozwiązuje w pełni problemu powtarzalności. Podobnie jak w przypadku dekodowania zachłannego, wygenerowany tekst brzmi sztucznie i mało naturalnie, co wynika z zupełnego braku słów o niższym prawdopodobieństwie wystąpienia.



Rysunek 5.7. Wyszukiwanie wiązkowe

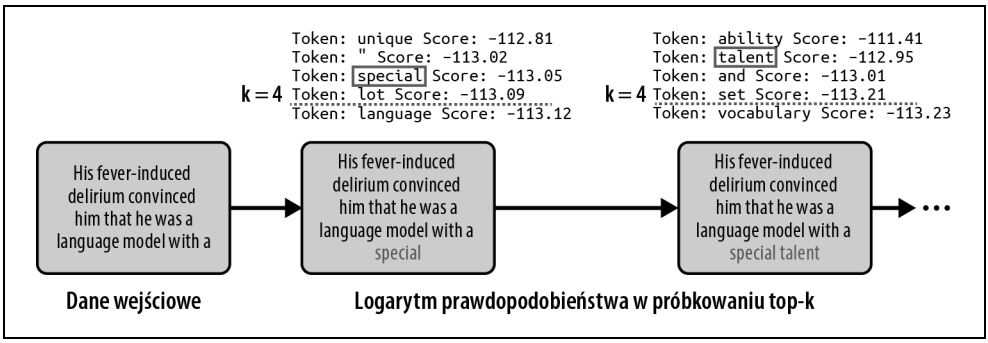
Aby rozwiązać te problemy, musimy wprowadzić element losowości i zacząć próbkować z rozkładu prawdopodobieństwa. Unikniemy w ten sposób sytuacji, w której za każdym razem generowane są dwa lub trzy najbardziej prawdopodobne tokeny.

Próbkowanie top-k

W próbkowaniu top-k model wybiera spośród *k* tokenów o najwyższym prawdopodobieństwie w rozkładzie wyjściowym. Masa prawdopodobieństwa jest redystrybuowana między *k* tokenów, a model losuje z tego nowego rozkładu, aby wygenerować kolejny token. Biblioteka `Hugging Face` udostępnia parametr `top_k` w funkcji `generate`:

```
output = model.generate(**inputs, max_new_tokens=50, do_sample=True, top_k=40)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

Na rysunku 5.8 został przedstawiony przykład próbkowania top-k z wykorzystaniem modelu FLAN-T5. Zapewnia to znaczną poprawę w porównaniu z dekodowaniem zachłannym lub wyszukiwaniem wiązkowym. Jednak próbkowanie top-k może prowadzić do problemów w przypadkach, gdy rozkład prawdopodobieństwa jest zdominowany przez kilka tokenów. Oznacza to, że w zbiorze top-k mogą zostać uwzględnione tokeny o bardzo niskim prawdopodobieństwie.



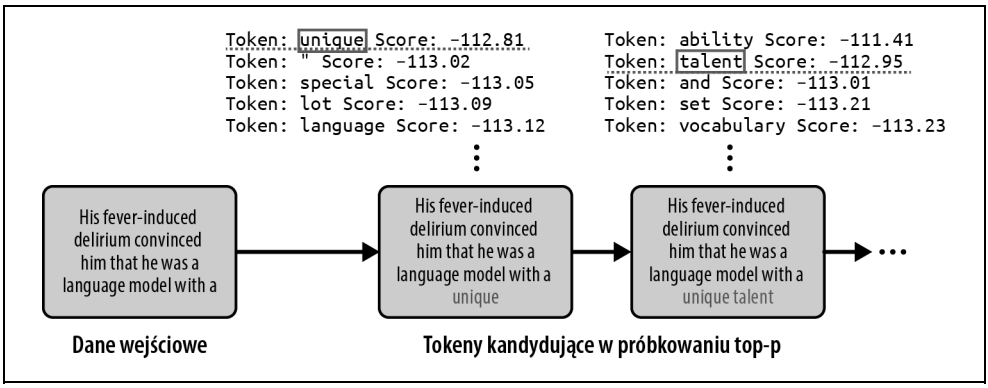
Rysunek 5.8. Próbkowanie top-k

Próbkowanie top-p

Próbkowanie top-p rozwiązuje problem z próbkowaniem top-k, wprowadzając dynamiczną liczbę kandydujących tokenów. Metoda top-p polega na wyborze najmniejszej liczby tokenów, których łączny rozkład prawdopodobieństwa przekracza zadaną wartość p. Oto jak można zaimplementować to z użyciem biblioteki `Hugging Face Transformers`:

```
output = model.generate(**inputs, max_new_tokens=50, top_p=0.9)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

Na rysunku 5.9 został przedstawiony przykład próbkowania top-p z wykorzystaniem modelu FLAN-T5. Próbkowanie top-p, zwane również próbkowaniem jądrowym, jest obecnie najpopularniejszą strategią próbkowania.



Rysunek 5.9. Próbkowanie top-p

Omówione dotychczas metody dekodowania działają szeregowo, co oznacza, że każdy token jest generowany pojedynczo, co za każdym razem wymaga pełnego przebiegu przez model. W zastosowaniach wrażliwych na opóźnienia jest to zbyt nieefektywne. W rozdziale 9. omawiam metody takie jak dekodowanie spekulacyjne, które mogą przyspieszyć proces dekodowania.

Wnioskowanie z użyciem LLM-ów

Teraz, gdy wiesz już, jak uzyskać dostęp do LLM-ów i jak je wczytywać, oraz rozumiesz proces dekodowania, możesz zacząć wykorzystywać je do rozwiązywania rzeczywistych zadań. Nazywamy to **wnioskowaniem z użyciem LLM-ów**.

Ćwiczenie

Jesteś muzykiem wyruszającym w trasę koncertową obejmującą siedem miast: Amsterdam, Warszawa, Hamburg, Barcelona, Delhi, Szanghaj i Toronto. Zapytaj model językowy, czy może zaproponować kolejność odwiedzania miast, która zapewni najkrótszy czas podróży. Aby rozwiązać to zadanie, wykorzystaj techniki i strategie formułowania promptów, które znasz z rozdziału 1.

Powtórz to dla kilku modeli językowych: modelu 3B, modelu 7B, modelu mającego co najmniej 30 mld parametrów oraz API zastrzeżonego LLM-a. Czy łatwo jest pokierować każdym modelem, żeby wykonał polecenie?

Ponadto repozytorium książki na GitHubie (*https://oreil.ly/llm-playbooks*) zawiera wiele przykładowych zadań, na których możesz przetestować swoje umiejętności formułowania promptów. Wypróbuj je i sprawdź, czy potrafisz skłonić modele językowe do udzielenia poprawnych odpowiedzi!

Zapewne zauważyłeś, że wyniki generowane przez modele językowe nie są spójne i czasami znacznie się różnią przy wielokrotnym generowaniu dla tego samego promptu. Jak wiesz z podrozdziału poświęconego dekodowaniu, jeśli nie używasz wyszukiwania zachłannego lub innego deterministycznego algorytmu, model językowy próbkuje z rozkładu prawdopodobieństwa tokenów.

Jednym ze sposobów na zwiększenie determinizmu generowania jest ustawienie temperatury na zero i utrzymywanie stałej wartości ziarna losowego dla próbkowania. Nawet wtedy możesz nie być w stanie zagwarantować tych samych (deterministycznych) wyników za każdym razem, gdy wyślesz do LLM-a ten sam prompt.

Źródła niedeterminizmu obejmują wykorzystanie wielowątkowości, błędy zaokrąglania liczb zmiennoprzecinkowych oraz stosowanie niektórych architektur modeli (wiadomo na przykład, że architektura Sparse MoE (*https://oreil.ly/pzchE*) generuje niedeterministyczne wyniki).

Zmniejszenie temperatury do zera lub bliskiej zeru wpływa na kreatywność modelu językowego i sprawia, że jego wyniki stają się bardziej przewidywalne, co może nie być odpowiednie w wielu zastosowaniach.

W środowiskach produkcyjnych, w których ważna jest niezawodność, należy przeprowadzać wiele generacji dla tych samych danych wejściowych i stosować techniki takie jak głosowanie większościowe lub heurystyki w celu wyboru właściwego wyniku. Jest to bardzo istotne ze względu na charakter procesu dekodowania; czasami mogą zostać wygenerowane niewłaściwe tokeny, a ponieważ każdy wygenerowany token jest funkcją tokenów wygenerowanych przed nim, błąd może się rozprzestrzeniać daleko w przód.

Ćwiczenie

Dla każdego z podanych ćwiczeń w podpowiadaniu przeprowadź wielokrotne generacje i sprawdź, jak zmieniają się wyniki. Czy głosowanie większościowe dobrze się sprawdza w wyborze poprawnego wyniku?

Spójność wewnętrzna (ang. *self-consistency*) (*https://oreil.ly/wEE8q*) to popularna technika podpowiadania, która wykorzystuje głosowanie większościowe w połączeniu z promptami typu CoT. W technice tej dodajemy do danych wejściowych podpowiedź CoT „Pomyślmy krok po kroku" i przeprowadzamy wiele generacji (ścieżek rozumowania). Następnie używamy głosowania większościowego do wyboru poprawnego wyniku.

Wyniki ustrukturyzowane

W niektórych przypadkach wyniki działania LLM-a powinny mieć format ustrukturyzowany, żeby można było wykorzystać je w innych systemach informatycznych. Łatwiej to jednak powiedzieć niż zrobić, obecne LLM-y nie są bowiem tak sterowalne, jak byśmy chcieli. Niektóre z nich potrafią być nadmiernie gadatliwe. Gdy poprosimy je o odpowiedź tak/nie, mogą odpowiedzieć: „Odpowiedzią na to pytanie jest »tak«.".

Jednym ze sposobów na uzyskanie ustrukturyzowanych wyników z LLM-a jest zdefiniowanie schematu JSON, dostarczenie go do modelu i poproszenie o generowanie wyników zgodnych z tym schematem. W przypadku większych modeli działa to prawie zawsze, z pewnymi błędami w strukturze JSON, które można wychwycić i obsłużyć.

Dla mniejszych modeli można wykorzystać biblioteki takie jak `Jsonformer` (*https://oreil.ly/aSc0f*). Przekazuje ona generowanie treści tokenów do LLM-a, ale samodzielnie wypełnia formularz JSON. `Jsonformer` jest zbudowany na bazie Hugging Face, więc obsługuje każdy model wspierany przez tę platformę.

Ćwiczenie

Wyodrębnij tekst z sekcji kariery ze strony Wikipedii aktora Andrew Garfielda (*https://oreil.ly/WXtgc*). Zaprojektuj schemat JSON z następującymi typami treści: współaktorzy, reżyser, rok i tytuł filmu. Użyj otwartego modelu językowego, aby wydobyć szczegóły dotyczące filmów z Garfieldem z nieustrukturyzowanego tekstu, a następnie zastosuj bibliotekę `Jsonformer` lub podobną, aby przedstawić je w ustrukturyzowanej formie. Czy jesteś w stanie uzyskać w pełni sformowane i dokładne dane wyjściowe JSON?

Bardziej zaawansowane ustrukturyzowane dane wyjściowe można uzyskać za pomocą bibliotek takich jak `LMQL` (*https://oreil.ly/LlkEj*) lub `Guidance` (*https://oreil.ly/cFe5s*). Biblioteki te zapewniają paradygmat programowania promptów i ułatwiają kontrolowaną generację.

Oto funkcje dostępne w tych bibliotekach:

Ograniczanie danych wyjściowych do skończonego zbioru tokenów

Jest to przydatne w problemach klasyfikacji, gdzie mamy skończony zbiór etykiet wyjściowych. Można na przykład ograniczyć dane wyjściowe do pozytywnych, negatywnych lub neutralnych w zadaniu z zakresu analizy odczuć.

Kontrolowanie formatu wyjściowego za pomocą wyrażeń regularnych

Można na przykład użyć wyrażeń regularnych do określenia niestandardowego formatu daty.

Kontrolowanie formatu wyjściowego za pomocą gramatyk bezkontekstowych (CFG)

CFG definiuje reguły, które muszą być spełnione przez generowane ciągi znaków. Więcej informacji na temat CFG można znaleźć na blogu Adityi (*https://oreil.ly/M00us*). Używając CFG, możemy efektywniej wykorzystywać modele językowe do wykonywania zadań tagowania sekwencji, takich jak rozpoznawanie nazwanych encji czy oznaczanie części mowy.

Ćwiczenie

Rozpoznawanie nazwanych encji (ang. *named entity recognition*, NER) to zadanie z zakresu sekwencyjnego tagowania polegające na oznaczaniu w tekście nazwanych encji, takich jak liczby, daty, miejsca, nazwy, organizacje itp. Na przykład dla zdania „Padma sprzedała 23 parasole w Gwatemali" oznakowane wyjście może mieć następującą formę:

Padma: PER

sprzedała:

23: NUM

parasole:

w:

Gwatemali: LOC

gdzie `PER` to tag dla osoby, `NUM` to tag dla liczb, a `LOC` to tag dla lokalizacji.

Aby wygenerować oznakowane wyjście w powyższym formacie, skorzystaj z biblioteki `Guidance` (*https://oreil.ly/R3XLQ*) i użyj wyrażenia CFG. Uruchom zadanie NER na stronie Wikipedii poświęconej letnim igrzyskom olimpijskim (*https://oreil.ly/tUrIt*). Do rozwiązania tego zadania użyj otwartego modelu językowego o wielkości 3B/7B.

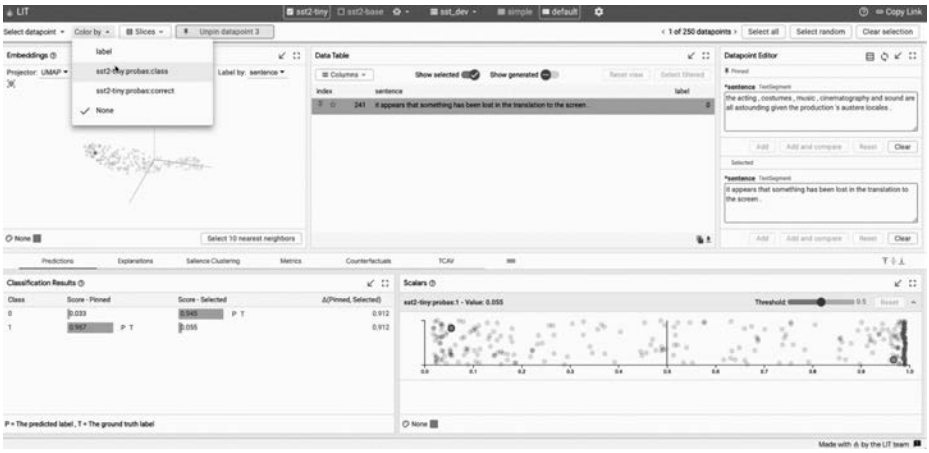
Debugowanie i interpretowanie modeli

Teraz, gdy umiesz już wczytywać LLM-y i używać ich do generowania tekstu, pora lepiej zrozumieć ich działanie i przeanalizować przypadki, w których zawodzą. Interpretowalność LLM-ów jest znacznie mniej rozwinięta niż w innych obszarach uczenia maszynowego. Możemy

jednak uzyskać częściową interpretowalność, badając zmiany wyników przy niewielkich modyfikacjach danych wejściowych oraz analizując pośrednie wyniki w trakcie przetwarzania danych przez architekturę transformera.

Otwartoźródłowe narzędzie LIT-NLP firmy Google (*https://oreil.ly/YFY4q*) to przydatne rozwiązanie, które umożliwia wizualizację działania modelu oraz obsługuje różne procesy debugowania.

Rysunek 5.10 przedstawia przykład działania narzędzia LIT-NLP, które zapewnia interpretowalność modelu T5 wykonującego zadanie z zakresu streszczania tekstu.



Rysunek 5.10. LIT-NLP

Oto funkcje LIT-NLP, które pomagają w debugowaniu modeli:

- wizualizacja mechanizmu uwagi;
- mapy istotności, pokazujące części danych wejściowych, na które model zwraca największą uwagę;
- wizualizacja osadzeń;
- analiza kontrfaktyczna, która pokazuje, jak zmienia się zachowanie modelu po zmodyfikowaniu danych wejściowych, na przykład po dodaniu lub usunięciu tokena.

Ćwiczenie

Korzystając ze zdań z kanadyjskiego zbioru danych z obrad parlamentarnych dostępnego w repozytorium książki na GitHubie (*https://oreil.ly/llm-playbooks*), sklasyfikuj zdania na podstawie ich tonu. Etykiety wyjściowe to: `supportive` (wspierający), `antagonistic` (antagonistyczny), `mournful` (żałobny), `celebratory` (radosny) i inny. Użyj promptów z kilkoma przykładami dla każdej etykiety. Wykorzystaj model Gemma od Google'a (dowolna wersja). Prawdopodobnie nie uzyskasz 100% skuteczności za pierwszym razem. Użyj narzędzia LIT-NLP do analizy błędów i sprawdź, czy możesz wykorzystać narzędzia do interpretacji, aby zebrać spostrzeżenia pozwalające ulepszyć model.

Więcej szczegółów na temat korzystania z LIT-NLP do analizy błędów znajdziesz w poradniku Google'a (*https://oreil.ly/zcsLu*) dotyczącym użycia LIT-NLP z modelem Gemma. Pokazano tam, jak znaleźć błędy w promptach z kilkoma przykładami poprzez analizę niepoprawnych wyników i obserwowanie, które części promptu miały największy wpływ na wyjście (istotność).

Interpretowalność mechanistyczna

Jak wyjaśniłem w rozdziale 2., najmniejszą jednostką LLM-a opartego na architekturze transformera jest neuron. Dlatego analiza zachowania pojedynczych neuronów w LLM-ie stanowi podstawowy krok w kierunku uczynienia tych modeli bardziej interpretowalnymi.

Jednakże badacze z firmy Anthropic zaobserwowali w swoich eksperymentach (*https://oreil.ly/hLdVN*), że pojedynczy neuron może być aktywowany przez wiele typów danych wejściowych. W związku z tym dokładny wkład danego neuronu nie jest do końca jasny. Naukowcy wprowadzili pojęcie cech będących liniowymi kombinacjami aktywacji wielu neuronów. Wykazali, że cechy te są bardziej interpretowalne niż pojedynczy neuron, ponieważ każda cecha jest aktywowana tylko przez jeden typ danych wejściowych. Niektóre cechy są aktywowane tylko przez pojedynczy token, podczas gdy inne reagują na szerszy zakres danych, na przykład kod źródłowy.

Więcej szczegółów można znaleźć w artykule firmy Anthropic na temat interpretowalności mechanistycznej (*https://oreil.ly/hLdVN*), w którym autorzy przeprowadzają eksperymenty na jednowarstwowym bloku transformera i identyfikują interesujące cechy.

Możesz dowiedzieć się więcej, korzystając z narzędzia do wizualizacji firmy Anthropic (*https://oreil.ly/2YvE6*), które zawiera tekstowe opisy tokenów aktywujących poszczególne neurony. Jako przykład autorzy pokazują (*https://oreil.ly/cE27M*), jak reaguje każdy neuron, gdy na wejście podawana jest treść książki *Alicja w Krainie Czarów*.

Podsumowanie

W tym rozdziale przyjrzeliśmy się różnym LLM-om i dostępnym opcjom. Wiesz już, jak określać kryteria najbardziej istotne w danym zadaniu i jak dobrać właściwy model. Omówiłem różne testy porównawcze modeli językowych i pokazałem, jak interpretować ich wyniki. Dowiedziałeś się, jak wczytywać modele i przeprowadzać wnioskowanie, oraz poznałeś efektywne strategie dekodowania. Na koniec zaprezentowałem narzędzia do interpretacji, takie jak LIT-NLP, które pomagają zrozumieć, co dzieje się „za kulisami" w architekturze transformera.

W następnym rozdziale nauczysz się aktualizować model, aby poprawić jego wydajność w interesujących Cię zadaniach. Przeprowadzę Cię przez pełny przykład dostrajania modelu i omówię decyzje związane z doborem hiperparametrów. Dowiesz się również, jak tworzyć zbiory danych treningowych do dostrajania.

To wyjątkowe opracowanie zawiera wszystkie ważne koncepcje w dziedzinie LLM!

Madhav Singhal, CEO, AutoComputer

Duże modele językowe przeniknęły do wielu dziedzin techniki — uważa się je za skuteczne narzędzia do rozwiązywania szerokiej gamy problemów. Coraz więcej przedsiębiorstw korzysta z ich potencjału w celu własnego rozwoju. Jednak przekształcenie prototypów w funkcjonalne aplikacje bywa złożone i skomplikowane.

W tej praktycznej książce opisano wszelkie niezbędne narzędzia, techniki i rozwiązania, których potrzebujesz do tworzenia użytecznych produktów wykorzystujących potęgę modeli językowych. Na początku zdobędziesz wiedzę o budowie modelu językowego. Następnie poznasz różne sposoby zastosowania modeli językowych, czy to poprzez bezpośrednie zapytania do modelu, czy też poprzez jego dostrajanie. Zrozumiesz ograniczenia LLM, takie jak halucynacje i problemy z rozumowaniem, a także dowiesz się, jak sobie z nimi poradzić. Znajdziesz tu również omówienie paradygmatów zastosowań, takich jak generowanie wspomagane wyszukiwaniem (RAG) czy agenty.

Z tą książką:

- przygotujesz zbiory danych do treningu i dostrajania modeli
- zrozumiesz architekturę transformera
- zaadaptujesz wstępnie wytrenowane modele do własnych potrzeb
- poznasz skuteczne techniki optymalizacji i adaptacji dziedzinowej
- nauczysz się integrować modele językowe z zewnętrznymi środowiskami i źródłami danych

Mistrzowski kurs budowania zaawansowanych systemów AI!

Jay Alammar, autor książek

Suhas Pai jest współzałożycielem i dyrektorem Hudson Labs, startupu z sektora AI i fintech wspieranego przez Y Combinator. Przyczynił się do rozwoju kilku modeli językowych o otwartym kodzie źródłowym. Współkierował grupą roboczą do spraw prywatności w projekcie BigScience, w ramach którego powstał model BLOOM.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-3111-4
9 788328 931114
Cena: 89,00 zł