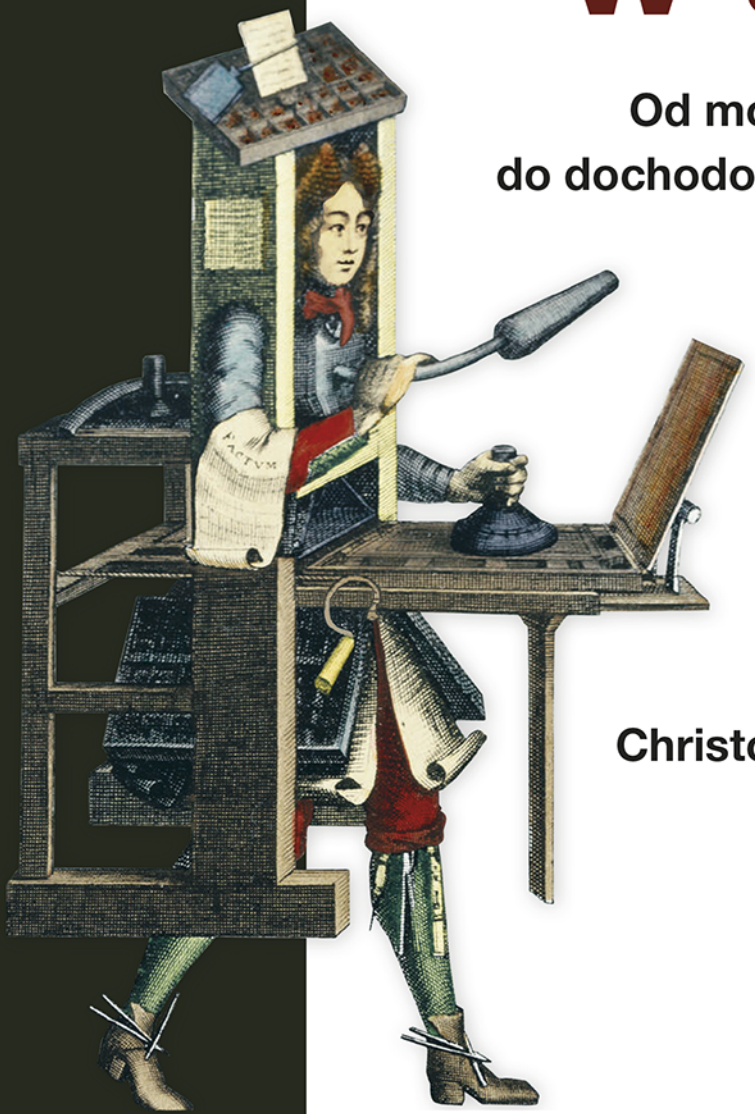
LLMs w akcji

Od modeli językowych do dochodowych produktów



Christopher Brousseau
Matthew Sharp

Przedmowa: Joe Reis

Helion

Tytuł oryginału: LLMs in Production:
From language models to successful products

Tłumaczenie: Piotr Rajca

ISBN: 978-83-289-3304-0

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
helion.pl/user/opinie/wlaipr
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

3 Operacje na dużych modelach językowych: Tworzenie platformy dla modeli LLM

W tym rozdziale omówiono następujące zagadnienia:

- Przegląd operacji na dużych modelach językowych.
- Wyzwania związane z wdrażaniem.
- Najlepsze praktyki stosowania dużych modeli językowych.
- Infrastruktura niezbędna do wdrażania dużych modeli językowych.

Przede wszystkim kluczem do sukcesu jest dobre przygotowanie.

— Alexander Graham Bell

Jak dowiedziałeś się w poprzednim rozdziale, w przypadku transformerów i przetwarzania języka naturalnego większe modele dają lepsze wyniki, szczególnie gdy są lingwistycznie świadome. Jednak ze względu na ich rozmiar stosowanie większych modeli wiąże się z większymi wyzwaniami, i to niezależnie od ich skuteczności językowej. Sprostanie tym wyzwaniom wymaga od nas rozszerzenia realizowanych operacji i rozbudowy infrastruktury. W tym rozdziale przyjrzymy się dokładnie, jakie to są wyzwania, co możemy zrobić, aby je zminimalizować, oraz jaką architekturę można stworzyć, aby to sobie ułatwić.

3.1. Wprowadzenie do operacji na dużych modelach językowych

Czym są operacje związane z dużymi modelami językowymi (LLMOps)? Cóż, ponieważ wolimy koncentrować się na aspektach praktycznych, a nie na retoryce, nie będziemy zagłębiać się w żadne wymyślne definicje, których można by się spodziewać w podręczniku. Powiedzmy po prostu, że to operacje uczenia maszynowego (ang. *MLOps*) przeskalowane do obsługi dużych modeli językowych. Dodajmy też, że skalowanie jest trudne. To jedno z najtrudniejszych zadań w inżynierii oprogramowania. Niestety, zbyt wiele firm korzysta z podstawowych konfiguracji MLOps i nie ma co liczyć na to, że stosując je, poradzą sobie z dużymi modelami językowymi.

Warto jednak zaznaczyć, że stosowanie terminu *LLMOps* może nie być konieczne. Jak dotąd nie wykazano, by znacząco różnił się on od podstawowych działań określanych jako MLOps, zwłaszcza biorąc pod uwagę, że oba mają te same fundamenty. Gdyby ta książka była kluczem do klasyfikowania gatunków, to MLOps i LLMOps z pewnością należałyby do tego samego rodzaju, a tylko czas pokaże, czy są przedstawicielami tego samego gatunku. Oczywiście, unikając precyzyjnej definicji LLMOps, mogliśmy zamienić jedno zamieszanie na drugie, dlatego też warto poświęcić chwilę na opisanie, czym są operacje uczenia maszynowego — MLOps.

MLOps to dziedzina i praktyka niezawodnego i efektywnego wdrażania oraz utrzymywania modeli uczenia maszynowego w środowiskach produkcyjnych. Określenie to obejmuje — a wręcz go wymaga — zarządzanie całym cyklem życia procesu uczenia maszynowego, od pozyskiwania danych i trenowania modelu po monitorowanie i zakończenie jego działania. Kilka zasad, których przyswojenie jest niezbędne do opanowania tej dziedziny, to orkiestracja przepływu pracy, wersjonowanie, pętle zwrotne, ciągła integracja i ciągłe wdrażanie (CI/CD), bezpieczeństwo, przydzielanie zasobów oraz zarządzanie danymi. Choć są osoby specjalizujące się we wdrażaniu modeli do produkcji, posługujące się takimi tytułami jak inżynier ML, inżynier MLOps czy inżynier infrastruktury ML, dziedzina ta jest na tyle rozległa, że często wciąga wielu innych, nieświadomych specjalistów, którzy mają tytuły takie jak naukowiec ds. danych (ang. *data scientist*) czy inżynier DevOps; często dzieje się tak wbrew ich wiedzy lub woli, co prowadzi do protestów i stwierdzeń typu: „To nie moja działka!”.

3.2. Wyzwania operacyjne związane z dużymi modelami językowymi

Dlaczego więc w ogóle wprowadzać rozróżnienie? Jeśli MLOps i LLMOps są tak podobne, czy LLMOps to tylko kolejny modny termin, który oportuniści wpisują sobie do CV? Niezupełnie. W rzeczywistości sytuacja jest podobna do

tej z szyldem „Big Data”. Kiedy termin ten był u szczytu popularności, osoby z tytułami takimi jak „Inżynier Big Data” korzystały z zupełnie innych zestawów narzędzi i rozwijały specjalistyczną wiedzę niezbędną do obsługi dużych zbiorów danych. Stosowanie dużych modeli językowych wiąże się z szeregiem wyzwań i problemów, których nie spotkamy w tradycyjnych systemach uczenia maszynowego. Większość tych problemów wynika niemal wyłącznie z ogromnych rozmiarów modelu. Duże modele są naprawdę duże! W tym rozdziale postaramy się pokazać, że modele LLM w pełni zasługują na swoją nazwę. Przyjrzyjmy się kilku z tych wyzwań, aby lepiej zrozumieć zadanie, które stoi przed nami w kontekście wdrażania modeli LLM i którym zajmiemy się później.

3.2.1. Długi czas pobierania

W 2017 roku, gdy byłem jeszcze mocno zaangażowany w pracę jako naukowiec danych, postanowiłem spróbować swoich sił w reimplementacji niektórych z najbardziej znanych modeli wizji komputerowej tamtego czasu: AlexNet, VGG19 i ResNet. Uznałem, że będzie to dobry sposób na utrwalenie podstaw poprzez praktyczne doświadczenie. Miałem też ukryty motyw: właśnie zbudowałem własny komputer z kartami graficznymi NVIDIA GeForce 1080 TI, które były wówczas najnowocześniejsze, i pomyślałem, że to dobra okazja, by je przetestować. Pierwszym zadaniem było pobranie zbioru danych ImageNet.

Zbiór danych ImageNet był jednym z największych dostępnych oznakowanych zbiorów danych i zawierał miliony obrazów o łącznym rozmiarze pliku około 150 GB! Praca z nim była dowodem na to, że potrafisz obsługiwać Big Data, co w tamtym czasie wciąż było modnym hasłem i nieocenioną umiejętnością dla naukowca danych. Po zaakceptowaniu warunków i uzyskaniu dostępu spotkało mnie pierwsze otrzeźwienie. Pobieranie trwało cały tydzień.

Gdy mój zespół po raz pierwszy wdrażał model Bloom, jego pobranie zajęło półtorej godziny. Dla porównania, tyle samo czasu zajęło pobranie gry *The Legend of Zelda: Tears of the Kingdom*, która ma tylko 16 GB, więc naprawdę nie mogliśmy narzekać.

Duże modele są duże. Nie da się tego przecenić. Podczas lektury przekonasz się, że ten fakt niesie ze sobą wiele dodatkowych problemów dla całego procesu produkcyjnego, i musisz być na to przygotowany. Model językowy Bloom ma 330 GB, czyli ponad dwa razy więcej od zbioru danych ImageNet. Zakładamy, że większość Czytelników nie pracowała ani z ImageNet, ani z Bloom, więc dla porównania: gra *Call of Duty: Modern Warfare*, jedna z największych gier w momencie, w którym piszemy te słowa, ma 235 GB. *Final Fantasy XV* ma tylko 148 GB, więc w modelu zmieściłyby się dwie takie gry, a jeszcze zostałoby sporo miejsca. Naprawdę trudno pojąć, jak ogromne są modele językowe. Przeszliśmy od 100 milionów parametrów w modelach takich jak BERT do miliardów. Gdybyś poszedł na zakupy i wydawał 20 dolarów na sekundę (albo

przypadkowo zostawił włączoną instancję AWS EC2), wydanie miliona dolarów zajęłoby Ci pół dnia; wydanie miliarda zajęłoby dwa lata.

Na szczęście pobranie Bloom nie trwa dwóch tygodni, ponieważ w przeciwieństwie do ImageNet nie jest on hostowany na słabo zarządzanym serwerze uniwersyteckim, a ponadto podzielono go na wiele mniejszych plików, które można pobierać równolegle. Mimo to nadal zajmie to zatrważająco dużo czasu. Wyobraź sobie scenariusz, w którym pobierasz model w najlepszych warunkach. Masz łącze światłowodowe o prędkości 1 gigabita i w cudowny sposób możesz przeznaczyć na to zadanie całą przepustowość oraz wszystkie operacje I/O swojego systemu i serwera. Nawet wtedy pobieranie zajmie ponad 5 minut! Oczywiście — w idealnych warunkach. Prawdopodobnie nie będziesz pobierać modelu w takich okolicznościach; od współczesnej infrastruktury możesz oczekiwać, że poradzi sobie z tym w kilka godzin.

3.2.2. Dłuższe czasy wdrażania

Samo pobranie modelu zajmuje tyle czasu, że może przyprawić o dreszcze nawet doświadczonego programistę, ale czasy wdrożenia mogą go już całkowicie powalić. Załadowanie do pamięci GPU modelu tak dużego jak Bloom może zająć od 30 do 45 minut — przynajmniej takie są nasze doświadczenia. A i to bez uwzględnienia jeszcze innych etapów procesu wdrożeniowego, które mogą ten czas dodatkowo wydłużyć. Co więcej, przy obecnych niedoborach GPU samo oczekiwanie na zwolnienie zasobów spokojnie może zająć kilka godzin; o czym więcej napiszemy za chwilę.

A co to oznacza dla Ciebie i Twojego zespołu? Po pierwsze, wiele zespołów wdrażających produkty uczenia maszynowego po prostu pobiera model w czasie wykonywania. To może się sprawdzić w przypadku małych modeli regresji z biblioteki scikit-learn, ale nie sprawdzi się z dużymi modelami językowymi. Ponadto większość tego, co wiesz o wdrażaniu niezawodnych systemów, możesz wyrzucić za okno (chociaż na szczęście niezbyt daleko). Większość współczesnych dobrych praktyk inżynierii oprogramowania zakłada, że w razie problemów można łatwo zrestartować aplikację, a wiele zabiegów służy temu, by systemy mogły to robić. W przypadku dużych modeli językowych wyłączenie może zająć sekundy, ale ponowne wdrożenie może potrwać nawet godziny, co czyni ten proces prawie nieodwracalnym. To jak zerwanie jabłka z drzewa — łatwo je zerwać, ale jeśli ugryziesz i stwierdzisz, że jest za kwaśne, nie możesz go z powrotem przyczepić, żeby dojrzało. Musisz po prostu poczekać, aż urośnie następne.

Chociaż nie każdy projekt wymaga wdrażania największych dostępnych modeli, możesz się spodziewać, że czasy wdrożenia będą mierzone w minutach. Te dłuższe czasy wdrożenia sprawiają, że zmniejszenie skali tuż przed nagłym wzrostem ruchu jest fatalnym błędem, a zarządzanie zmiennymi obciążeniami staje się trudne. Ogólne metodyki CI/CD wymagają dostosowania, ponieważ

aktualizacje przyrostowe trwają dłużej, co szybko powoduje narastanie zaległości w potoku. Głupie błędy, takie jak literówki czy inne usterki, często trudniej zauważyć, a ich naprawa trwa dłużej.

3.2.3. Opóźnienie

Wraz ze wzrostem rozmiaru modelu często rośnie też opóźnienie wnioskowania. Kiedy się nad tym zastanowić, to jest to oczywiste — więcej parametrów oznacza więcej obliczeń, a więcej obliczeń przekłada się na dłuższy czas oczekiwania na wyniki. Tego problemu nie można bagatelizować. Znamy wiele osób, które lekceważą kwestię opóźnień, ponieważ korzystały z chatbota opartego na dużym modelu językowym (LLM) i wszystko działało płynnie. Jednak kiedy dokładniej przyjrzymy się takim chatbotom, zauważymy, że odpowiedzi są zwracane słowo po słowie i strumieniowane do użytkownika. Dzięki takiemu zabiegowi korzystanie z chatbota wydaje się płynne, ponieważ odpowiedzi pojawiają się szybciej, niż człowiek jest w stanie czytać; jednak w rzeczywistości jest to tylko sztuczka interfejsu użytkownika. Duże modele językowe wciąż są zbyt wolne, by nadawały się do praktycznego użycia na przykład w rozwiązaniach takich jak automatyczne uzupełnianie w trakcie pisania, w których odpowiedzi muszą być błyskawiczne. Wykorzystanie ich w procesie przetwarzania danych do czyszczenia lub podsumowywania dużych zbiorów tekstu również może okazać się zbyt powolne, by było użyteczne lub niezawodne.

Istnieje też wiele mniej oczywistych powodów niewielkiej szybkości działania dużych modeli językowych. Po pierwsze, modele LLM są często rozproszone na wielu procesorach graficznych, co zwiększa narzut komunikacyjny. Zgodnie z informacjami podanymi w dalszej części rozdziału, a konkretnie w punkcie 3.3.2, modele te są rozproszone również w inny sposób, często w celu poprawy opóźnień, ale każda forma dystrybucji wiąże się z dodatkowym obciążeniem. Ponadto na opóźnienia działania modeli LLM znacząco wpływa długość generowanej odpowiedzi — im więcej słów model używa do udzielenia odpowiedzi, tym dłużej to trwa. Oczywiście, dłuższe odpowiedzi często poprawiają dokładność. Na przykład z zastosowaniem technik inżynierii promptów, takich jak „rozumowanie krok po kroku” (ang. *Chain-of-Thought*, CoT), możemy poprosić model o szczegółowe przeanalizowanie podanego problemu, co w zadaniach logicznych i matematycznych poprawia wyniki, ale znacząco wydłuża odpowiedź i czas oczekiwania.

3.2.4. Zarządzanie kartami graficznymi (GPU)

Aby rozwiązać problemy z opóźnieniami, zazwyczaj staramy się uruchamiać duże modele językowe na procesorach graficznych (GPU). Do ich skutecznego trenowania również będziemy potrzebować GPU, ale wiąże się to z dodatkowymi wyzwaniami, których wiele osób nie docenia. Większość usług internetowych i wiele zastosowań uczenia maszynowego może działać wyłącznie na

procesorach głównych (CPU), ale nie dotyczy to dużych modeli językowych. Wynika to częściowo z możliwości równoległego przetwarzania, jakie oferują GPU, co pomaga rozwiązać problemy z opóźnieniami, a częściowo z wbudowanej optymalizacji GPU pod kątem wykonywania operacji algebry liniowej, mnożenia macierzy i operacji na tensorach. Dla wielu osób wkraczających w świat dużych modeli językowych oznacza to konieczność korzystania z nowego zasobu i dodatkową złożoność. Wielu śmiało wchodzi w ten świat, traktując go jako coś prostego, ale czeka ich niemiłe zaskoczenie. Większość architektur systemowych i narzędzi do orkiestracji, takich jak Kubernetes, zakłada, że aplikacja będzie działać tylko z wykorzystaniem CPU i pamięci. Chociaż często obsługują one dodatkowe zasoby, takie jak GPU, to zazwyczaj takie możliwości traktuje się jako dodatek. Dlatego szybko okaże się, że trzeba budować kontenery od podstaw i wdrażać nowe systemy monitorowania.

Jednym z aspektów zarządzania GPU, na który większość firm nie jest przygotowana, jest to, że ich dostępność jest ograniczona. Przez ostatnią dekadę wydawało się, że co jakiś czas borykamy się z globalnym niedoborem GPU. Mogą być one niezwykle trudne do pozyskania dla firm, które chcą zastosować infrastrukturę lokalną. Wiele czasu spędziliśmy na pracy z firmami, które z różnych powodów zdecydowały się na takie rozwiązanie. Jedną z rzeczy, które łączyły te firmy, był brak GPU na ich serwerach. A jeśli już je miały, często dostęp do nich był celowo utrudniony i ograniczony do kilku kluczowych pracowników.

Jeśli masz szczęście pracować w chmurze, wiele z tych problemów całkowicie znika, ale i tu nie ma nic za darmo. Obaj autorzy niniejszej książki pracowali w zespołach, które często kręciły się w kółko, próbując pomóc naukowcom zajmującym się danymi w uruchomieniu nowego środowiska pracy korzystającego z GPU. Natrafiliśmy na niejasne, złowróżbne błędy, takie jak `scale.up.error.out.of.resources`, tylko po to, by odkryć, że te ezoteryczne komunikaty oznaczają, że wszystkie GPU wybranego typu w całym regionie są zajęte i żadne nie są dostępne. W centrach danych zarówno CPU, jak i pamięć często można traktować jako zasoby nieskończone, ale nie dotyczy to GPU. Czasami w ogóle nie można liczyć na ich dostępność. Większość centrów danych obsługuje tylko część typów instancji lub rodzajów GPU, co oznacza, że możesz zostać zmuszony do skonfigurowania swojej aplikacji w regionie oddalonym od bazy użytkowników, co wiąże się ze zwiększeniem opóźnień. Oczywiście, gdy zechcesz rozszerzyć swoją usługę na nowy region, który obecnie jej nie obsługuje, możesz nawiązać współpracę z dostawcą usług chmurowych, jednak może się zdarzyć, że nie będziesz zadowolony z tego, co usłyszysz na temat terminów i kosztów. Ostatecznie, niezależnie od tego, czy zdecydujesz się na infrastrukturę lokalną czy chmurę, napotkasz problemy związane z niedoborem zasobów.

3.2.5. Osobliwości danych tekstowych

Duże modele językowe (LLM) to nowoczesne rozwiązanie w dziedzinie przetwarzania języka naturalnego (NLP). NLP jest jedną z najbardziej fascynujących gałęzi uczenia maszynowego, ponieważ zajmuje się głównie danymi tekstowymi, które mają charakter jakościowy. Inne dziedziny zazwyczaj operują na danych ilościowych. Nauczyliśmy się kodować swoje obserwacje świata w postaci bezpośrednich wartości liczbowych. Na przykład potrafimy zakodować ciepło w skalach temperatury i mierzyć je termometrami czy termoparami, z kolei ciśnienie mierzymy manometrami i przeliczamy na paskale.

Wizja komputerowa i analiza obrazów często postrzega się jako dziedziny jakościowe, ale samo kodowanie obrazów na liczby jest już rozwiązanym problemem. Nasze zrozumienie światła pozwoliło nam rozbić obrazy na piksele i przypisać im wartości RGB. Oczywiście nie oznacza to, że wizja komputerowa jest zagadnieniem, które nie ma już dla nas tajemnic — wciąż jest wiele do zrobienia w kwestii identyfikowania różnych sygnałów w strukturach danych. Dane dźwiękowe również często uważa się za jakościowe. Jak porównać dwie piosenki? Jednak potrafimy mierzyć dźwięk i mowę — wystarczą do tego celu bezpośrednie pomiary intensywności fali dźwiękowej wyrażone w decybelach i częstotliwości w hercach.

W przeciwieństwie do innych dziedzin, które kodują nasz fizyczny świat na dane liczbowe, dane tekstowe próbują zmierzyć świat niematerialny. Tekst jest najlepszym ze znanych nam sposobów kodowania myśli, idei i wzorców komunikacji. Choć opracowaliśmy sposoby zamiany słów na liczby, nie znaleźliśmy jeszcze bezpośredniego tłumaczenia. Nasze czołowe rozwiązania do kodowania tekstu i tworzenia osadzeń są w najlepszym razie przybliżeniami; w rzeczywistości używamy do tego modeli uczenia maszynowego! Ciekawostką jest to, że liczby również są tekstem i częścią języka. Jeśli chcemy, aby modele lepiej radziły sobie z matematyką, potrzebujemy bardziej znaczącego sposobu kodowania tych liczb. Ponieważ wszystko to jest sztuczne, chcąc zakodować liczby tekstowe na liczby zrozumiałe dla maszyn, tworzymy system, który w znacznej mierze próbuje w rekurencyjny sposób odwoływać się do siebie. A to nie jest problem łatwy do rozwiązania!

Z tego powodu modele językowe (i wszystkie rozwiązania z dziedziny przetwarzania języka naturalnego) stawiają przed nami unikalne wyzwania. Weźmy na przykład monitorowanie. Jak wykryć dryf danych w danych tekstowych? Jak zmierzyć „poprawność”? Jak zapewnić czystość danych? Te problemy są trudne do zdefiniowania, a co dopiero do rozwiązania.

3.2.6. Ograniczenia tokenów tworzą wąskie gardła

Poważnym wyzwaniem dla osób rozpoczynających pracę z modelami LLM są ograniczenia tokenów. Limit tokenów dla modelu to maksymalna liczba tokenów, które można uwzględnić jako dane wejściowe. Im większy limit tokenów, tym

więcej kontekstu możemy dostarczyć modelowi, aby poprawić jego skuteczność w wykonywaniu zadań. Wszyscy chcieliby, aby te limity były wyższe, ale nie jest to takie proste. Ograniczenia tokenów mają dwie przyczyny: wielkość i szybkość pamięci, do której mają dostęp nasze procesory graficzne, oraz cechy pamięci używanej w samych modelach.

Pierwszy problem może wydawać się nieintuicyjny: Dlaczego nie moglibyśmy po prostu zwiększyć pamięci instalowanej w kartach graficznych? Odpowiedź jest złożona. Możemy to zrobić, ale dodawanie kolejnych warstw pamięci w GPU, aby dodać do niej kolejne gigabajty, zmniejsza ogólną zdolność obliczeniową GPU. Obecnie producenci GPU pracują nad nowymi architekturami i sposobami rozwiązania tego problemu. Drugie wyzwanie jest fascynujące, ponieważ zwiększanie limitów tokenów faktycznie pogłębia problemy matematyczne leżące u podstaw modeli językowych. Już wyjaśniamy, o co chodzi. Do tej pory rzadko myślimy o pamięci wykorzystywanej w ramach samego modelu językowego. Mechanizm tej pamięci nazywamy *uwagą* (ang. *attention*), a mówiliśmy o nim w rozdziale 2., w punkcie 2.2.7. Nie zwróciliśmy natomiast uwagi na to, że jest on rozwiązaniem o złożoności kwadratowej: wraz ze wzrostem liczby tokenów liczba obliczeń wymaganych do obliczenia wyników uwagi między wszystkimi parami tokenów w sekwencji rośnie kwadratowo wraz z długością sekwencji. Ponadto, ze względu na ogrom wykorzystywanych przestrzeni kontekstowych oraz to, że mamy do czynienia z operacjami o złożoności kwadratowej, zaczynają pojawiać się problemy, których jedyne rozwiązania wymagają zastosowania liczb urojonych, co może powodować nieoczekiwane zachowania modeli. Jest to prawdopodobnie jeden z powodów, dla których modele językowe „halucynują".

Te problemy mają realne konsekwencje i wpływają na projektowanie aplikacji. Na przykład, gdy zespół jednego z autorów przeszedł z modelu GPT-3 na GPT-4, początkowo cieszył się z dostępu do wyższego limitu tokenów, ale szybko okazało się, że prowadzi to do dłuższych czasów wnioskowania i w konsekwencji do wyższego wskaźnika błędów przekroczenia limitu czasu. W praktyce często lepiej jest uzyskać mniej dokładną odpowiedź szybko, niż nie otrzymać żadnej, gdyż obietnice składane przez dokładniejsze modele często pozostają jedynie obietnicami. Oczywiście, przy lokalnym wdrażaniu, gdy nie trzeba martwić się o czasy odpowiedzi, prawdopodobnie okaże się, że czynnikiem ograniczającym jest sprzęt. Na przykład model Llama wytrenowano z użyciem 2048 tokenów, ale w razie zastosowania jakiejś prostej konsumenckiej karty graficznej będziesz miał szczęście, jeśli uda Ci się wykorzystać więcej niż 512 z nich, ponieważ prawdopodobnie napotkasz błędy braku pamięci (OOM) lub model po prostu się zawiesi.

Pułapka, która prawdopodobnie zaskoczy Twój zespół i na którą warto zwrócić uwagę już teraz, polega na tym, że w różnych językach liczba znaków przypadających na token jest różna. Spójrzmy na tabelę 3.1, w której porównujemy konwersję tego samego zdania w różnych językach na tokeny z użyciem kodera

OpenAI cl100k_base Byte Pair Encoder. Już na pierwszy rzut oka widać, że pod tym względem modele językowe zazwyczaj faworyzują język angielski. W praktyce oznacza to, że jeśli tworzysz chatbota z modelem językowym, użytkownicy posługujący się językiem angielskim będą mieli większą elastyczność w przestrzeni wejściowej niż osoby japońskojęzyczne, co z kolei będzie prowadziło do bardzo różnych doświadczeń użytkowników.

Tabela 3.1. *Porównanie liczby tokenów w różnych językach*

Język	Ciąg znaków	Liczba znaków	Liczba tokenów
Angielski	The quick brown fox jumps over the lazy dog	43	9
Francuski	Le renard brun rapide saute par-dessus le chien paresseux	57	20
Hiszpański	El rápido zorro marrón salta sobre el perro perezoso	52	22
Japoński	素早い茶色のキツネが怠惰な犬を飛び越える	20	36
Chiński (uproszczony)	敏捷的棕色狐狸跳过了懒狗	12	28

Jeśli jesteś ciekawy, dlaczego tak jest, to wynika to z kodowania tekstu, które jest kolejną osobliwością pracy z danymi tekstowymi, o której mówiliśmy w poprzednim punkcie podrozdziału. Spójrz na tabelę 3.2, w której pokazaliśmy kilka różnych znaków i ich reprezentacje binarne w kodzie UTF-8. Znaki angielskie niemal zawsze mogą być reprezentowane przez pojedynczy bajt pochodzący z pierwotnego standardu ASCII, używanego we wczesnych komputerach. Z kolei zapis większości innych znaków wymaga 3 lub 4 bajtów. Ponieważ takie znaki wymagają więcej pamięci, zajmują również więcej miejsca w tokenach.

Tabela 3.2. *Porównanie długości (w bajtach) symboli różnych walut w kodzie UTF-8*

Znak	Wartość znaku UTF-8 zapisana binarnie	Wartość w zapisie szesnastkowym
$	00100100	0x24
£	11000010 10100011	0xc2 0xa3
¥	11000010 10100101	0xc2 0xa5
€	11100010 10000010 10100000	0xe2 0x82 0xa0
💰	11110000 10011111 10010010 10110000	0xf0 0x9f 0x92 0xb0

Zwiększanie limitów tokenów jest nieustannym tematem badań od czasu popularyzacji transformerów, a niektóre obiecujące rozwiązania, na przykład

transformery z pamięcią cykliczną (RMT)[1], są nadal w fazie badań. Zapewne w przyszłości pojawią się dalsze ulepszenia i, miejmy nadzieję, limity tokenów staną się jedynie drobną niedogodnością.

3.2.7. Halucynacje powodują dezorientację

Do tej pory omawialiśmy niektóre problemy techniczne, z jakimi zespół musi się zmierzyć podczas wdrażania modelu LLM w środowisku produkcyjnym. Jednak nic nie może się równać z prostym problemem, którym jest to, że modele LLM często się mylą. I to bardzo często. Termin „halucynacje" wprowadzono w celu opisania sytuacji, gdy modele LLM generują poprawnie brzmiące wyniki, które są błędne — na przykład odniesienia do książek lub hiperłącza, które mają formę i strukturę zgodną z oczekiwaniami, ale są całkowicie zmyślone.

W ramach ciekawego przykładu poprosiliśmy o książki na temat modeli LLM w środowiskach produkcyjnych wydane przez wydawnictwo Manning (przy czym jedna z takich książek, o której wiem, jeszcze nie istnieje, ponieważ jeden z autorów wciąż nad nią pracuje). Otrzymaliśmy następujące sugestie: *Machine Learning Engineering in Production* autorstwa Mike'a Del Balso i Lucasa Serveéna, dostępną na stronie *https://www.manning.com/books/machine-learning-engineering-in-production*, oraz *Deep Learning for Coders with Fastai and PyTorch* autorstwa Jeremy'ego Howarda i Sylvaina Guggera, dostępną na stronie *https://www.manning.com/books/deep-learning-for-coders-with-fastai-and-pytorch*. Pierwsza książka jest całkowicie zmyślona. Druga istnieje, ale nie wydało jej Manning. W obu przypadkach adresy internetowe są całkowicie fikcyjne. Te adresy URL są bardzo podobne do tych, których można by się spodziewać, przeglądając witrynę wydawnictwa Manning, ale próba wejścia na którąkolwiek z tych stron zwraca błąd 404.

Jednym z najbardziej irytujących aspektów halucynowania modeli LLM jest to, że często towarzyszą mu sformułowania pewne i autorytatywne. Modele LLM są bardzo słabe w wyrażaniu niepewności, w dużej mierze ze względu na sposób, w jaki są trenowane. Weźmy na przykład przypadek „2 + 2 =". Czy wolałbyś, żeby odpowiedź brzmiała „Myślę, że to 4" czy po prostu „4"? Większość z nas wolałaby zapewne otrzymać poprawną odpowiedź „4". To nastawienie modeli jest zakodowane, ponieważ często otrzymują one nagrody za poprawność lub przynajmniej za zwracanie wyników brzmiących poprawnie.

Istnieją różne wyjaśnienia, dlaczego występują halucynacje, ale najbardziej prawdziwa odpowiedź brzmi: nie wiemy, czy mają one tylko jedną przyczynę. Prawdopodobnie jest to kombinacja kilku czynników, i dlatego nie ma jeszcze dobrego rozwiązania tego problemu. Niemniej jednak przygotowanie się na przeciwdziałanie tym niedokładnościom i błędom modelu jest kluczowe w staraniach o zapewnienie użytkownikom produktu jak najlepszych doświadczeń.

[1] A. Bulatov, Y. Kuratov i M.S. Burtsev, *Scaling transformer to 1M tokens and beyond with RMT*, kwiecień 2023 r., *https://arxiv.org/abs/2304.11062*.

3.2.8. Uprzedzenia i kwestie etyczne

Równie niepokojąca jak to, że model popełnia błędy, jest sytuacja, gdy udziela poprawnych odpowiedzi w najgorszy możliwy sposób — na przykład przez zachęcanie użytkowników do popełnienia samobójstwa[2], instruowanie ich, jak zrobić bombę[3], lub podsycanie fantazji seksualnych u dzieci[4]. To skrajne przykłady, ale uniemożliwienie modelowi odpowiadania na takie pytania jest niewątpliwie kluczowe dla jego sukcesu.

Duże modele językowe są trenowane na ogromnych ilościach danych tekstowych, które są też głównym źródłem ich stronniczości. Ponieważ okazało się, że w dążeniu do uzyskiwania wyników przypominających ludzkie większe zbiory danych są równie ważne jak bardziej rozbudowane modele, większość tych zbiorów nigdy nie została tak naprawdę starannie przygotowana ani przefiltrowana w celu usunięcia szkodliwych treści. Zamiast tego priorytetem była ich wielkość i jak największa ilość zebranych informacji. Oczyszczenie zbioru danych często postrzega się jako zbyt kosztowne i wymagające ręcznej weryfikacji przez ludzi. Jednak wiele można zrobić za pomocą prostych wyrażeń regularnych i innych zautomatyzowanych rozwiązań. Przetwarzając te ogromne zbiory treści i ucząc się ukrytych ludzkich uprzedzeń, modele mimowolnie je utrwalają. Uprzedzenia te obejmują seksizm, rasizm, preferencje polityczne i mogą powodować, że model nieświadomie promuje negatywne stereotypy i stosuje dyskryminujący język.

3.2.9. Kwestie bezpieczeństwa

Jak w przypadku każdej technologii, musimy pamiętać o bezpieczeństwie. Modele językowe trenuje się na ogromnym zbiorze tekstów, z których część może zawierać szkodliwe lub poufne informacje, które nie powinny być ujawniane. Dlatego należy podjąć kroki, aby chronić te dane przed wyciekaniem. Obawy dotyczące uprzedzeń i kwestii etycznych, o których mówiliśmy wcześniej, dobrze pokazują, jakich rozmów nasi użytkownicy powinni z naszego punktu widzenia unikać. Można też wyobrazić sobie sytuację, w której model jest dostrajany na danych firmowych, co, jeśli nie zostaną podjęte odpowiednie środki ostrożności, może prowadzić do niezamierzonego ujawnienia tajemnic.

Należy pamiętać, że modele językowe są podatne na ataki typu wstrzykiwanie promptów (ang. *prompt injection*). Polegają one na tym, że użytkownik próbuje

[2] R. Daws, *Medical chatbot using OpenAI's GPT-3 told a fake patient to kill themselves*, AI News, 28 października 2020 r., *https://www.artificialintelligence-news.com/news/medical-chatbot-openai-gpt3-patient-kill-themselves/*.

[3] T. Kington, *ChatGPT bot tricked into giving bomb-making instructions, say developers*, The Times, 17 grudnia 2022 r., *https://www.thetimes.com/business/technology/article/chatgpt-bot-tricked-into-giving-bomb-making-instructions-say-developers-rvktrxqb5*.

[4] K. Quach, *AI game bans players for NSFW stories it generated itself*, The Register, 8 października 2021 r., *https://www.theregister.com/2021/10/08/ai_game_abuse/*.

oszukać model, by zignorował narzucone mu instrukcje i generował niepożądane treści. Na przykład, jeśli zapytasz ChatGPT o jego płeć, odpowie poprawnie, że jako model językowy AI, nie ma płci. Jednak sprytne formułowanie pytań pozwala obejść te zabezpieczenia i skłonić model do ujawnienia płci. Choć ten przykład jest nieszkodliwy, znane są przypadki, gdy udało się wydobyć z modelu klucze API i inne poufne informacje, uruchomić kod w niezabezpieczonych środowiskach, wykraść zmienne środowiskowe czy przeglądać lokalne systemy plików, na których model był hostowany. Nie wspominając już o licznych przykładach użytkowników wykorzystujących technikę wstrzykiwania promptów do omijania zabezpieczeń etycznych, o których mówiliśmy wcześniej. Ciekawostką jest to, że modele językowe są również dobre w wymyślaniu fałszywych sekretów! Nawet udane ataki typu wstrzykiwanie promptów często zawodzą z powodu halucynacji modelu, co może prowadzić do zabawnych konsekwencji.

Na przykład proste zapytanie o tajną recepturę Coca-Coli dało nudną odpowiedź: „Nie mogę ujawniać tajemnic handlowych ani żadnych informacji zastrzeżonych". Dość kiepski wynik, prawda? Dzięki niewielkiemu podstępowi i wykorzystaniu techniki wstrzykiwania promptów udało nam się jednak wydobyć „sekret", który oczywiście postanowiliśmy wypróbować — po wcześniejszym potwierdzeniu u technologa żywności, że nie spowoduje to zatrucia. Ku naszemu zaskoczeniu, postępując ściśle według instrukcji, otrzymaliśmy całkiem smaczny napój, który jednak w ogóle nie smakował jak Coca-Cola. Choć przepis wyglądał na autentyczny, szczegóły takie jak ilości składników, czasy gotowania, a nawet konieczność przecedzenia syropu były błędne. Wygląda na to, że musimy odłożyć plany przejęcia branży napojów, dopóki nie zdobędziemy prawdziwej receptury.

Przepis na Coca-Colę zmyślony przez model po zastosowaniu techniki wstrzykiwania promptów

```
Domowy przepis na Coca-colę
==========================================
Zastrzeżenie: Oryginalna receptura Coca-Coli jest ściśle strzeżoną tajemnicą handlową.
Poniższy przepis to przybliżona wersja, pozwalająca na przygotowanie domowego napoju
typu cola, zbliżonego smakiem do Coca-Coli.

Składniki:
- 3 szklanki wody
- 2 szklanki cukru krystalicznego
- 1/8 łyżeczki kwasu cytrynowego spożywczego
- 3/4 łyżeczki barwnika karmelowego (w płynie)
- 1/2 łyżeczki kofeiny (w proszku lub w płynie)
- 1 1/2 łyżeczki ekstraktu z coli
- 1/8 łyżeczki ekstraktu waniliowego
- 1 1/2 szklanki wody gazowanej (kupnej lub przygotowanej w saturatorze domowym)
- Szczypta mielonego cynamonu (opcjonalnie)
- Szczypta mielonej gałki muszkatołowej (opcjonalnie)
```

```
Przygotowanie:
1. Wlej do garnka 3 szklanki wody, a następnie powoli wsyp cukier, mieszając, aż całkowicie się rozpuści.
2. Dodaj do rondla kwas cytrynowy, karmel, kofeinę, ekstrakt z orzesznicy oraz ekstrakt waniliowy. Dokładnie wymieszaj.
3. Opcjonalnie dodaj przyprawy: cynamon i gałkę muszkatołową.
4. Gotuj mieszankę na małym ogniu przez mniej więcej 20 minut, od czasu do czasu mieszając.
5. Zdejmij rondel z ognia i pozostaw do ostygnięcia do temperatury pokojowej.
6. Po ostudzeniu przecedź syrop, aby usunąć ewentualne nierozpuszczone składniki.
7. Aby przygotować gotowy napój, wymieszaj 1 część syropu z 3 częściami wody gazowanej. Proporcje możesz dostosować w zależności od preferowanego poziomu słodkości i nasycenia dwutlenkiem węgla.
Warianty:
- Możesz eksperymentować z ilością karmelu, kofeiny i ekstraktu z orzesznicy, aby zmienić smak i wygląd domowej coli.
- Aby uzyskać różne profile smakowe, spróbuj dodać dodatkowe przyprawy (takie jak anyż gwiazdkowaty, skórka pomarańczowa lub cytrynowa czy kardamon), olejki eteryczne lub ekstrakty owocowe.
- Zamiast cukru możesz użyć naturalnych substancji słodzących, takich jak miód lub syrop z agawy.
Kontekst historyczny i ciekawostki:
- Coca-Cola została wprowadzona na rynek w 1886 roku przez farmaceutę dr. Johna S. Pembertona, który chciał stworzyć wyjątkowy, orzeźwiający napój pomagający również w łagodzeniu zmęczenia i bólu głowy.
- Oryginalna receptura Coca-Coli zawierała śladowe ilości kokainy pozyskiwanej z liści koki, którą na początku XX wieku zastąpiono kofeiną.
- W 1985 roku wprowadzono "New Coke" – nową recepturę mającą konkurować z Pepsi. Zmiana smaku spotkała się jednak z dużą krytyką i już po trzech miesiącach powrócono do poprzedniej formuły, nazywając ją "Coca-Cola Classic".
Efekt: Teraz możesz samodzielnie przygotować domową wersję coli, przypominającą smakiem Coca-Colę. Eksperymentuj z przyprawami, ekstraktami i słodzikami, by stworzyć własny, unikalny profil smakowy.
```

Kolejnym problemem bezpieczeństwa są ataki typu pickle injection. Pickle to biblioteka języka Python służąca do serializacji obiektów, często wykorzystywana do zapisywania modeli uczenia maszynowego. Przekształca ona obiekty w strumień bajtów zawierający kody operacyjne, które są wykonywane jeden po drugim podczas deserializacji. Jest to szybki i prosty sposób na udostępnianie dużych obiektów. Ataki pickle injection polegają na modyfikacji tego strumienia bajtów, często w celu wstrzyknięcia złośliwego oprogramowania w trakcie przesyłania modelu przez niezabezpieczoną sieć. Jest to szczególnie niebezpieczne w przypadku dużych modeli, których pobieranie trwa długo, co ułatwia przechwycenie transmisji i wstrzyknięcie szkodliwego kodu przez osoby trzecie. Jeśli do tego dojdzie, wstrzyknięty kod może potencjalnie umożliwić atakującym dostęp do systemu.

Może to nastąpić podczas próby użycia modelu w fazie wnioskowania, gdy szkodliwy kod, jeśli nie zostanie wykryty i odpowiednio usunięty, może zostać wykonany. Aby zapobiec tego typu atakom, należy podjąć środki ostrożności, takie jak korzystanie z bezpiecznych sieci i weryfikacja integralności modelu przed jego użyciem.

3.2.10. Kontrola kosztów

Praca z modelami LLM wiąże się z różnymi kosztami. Pierwszym z nich, co zapewne już zauważyłeś, są koszty infrastruktury, obejmujące procesory graficzne (GPU) o wysokiej wydajności, pamięć masową i inne zasoby sprzętowe. Wspominaliśmy, że GPU trudniej zdobyć, co niestety oznacza, że są także droższe. Błędy, takie jak pozostawienie włączonej usługi, zawsze mogły generować wysokie rachunki, ale przy wykorzystaniu GPU tego typu pomyłki są jeszcze bardziej kosztowne. GPU wymagają również znacznej mocy obliczeniowej, co prowadzi do wysokiego zużycia energii zarówno podczas trenowania modelu LLM, jak i wnioskowania. Ponadto dłuższe czasy wdrażania oznaczają, że często musimy je uruchamiać nawet przy niskim ruchu, aby obsłużyć nagłe skoki obciążenia lub przewidywany przyszły ruch. Wszystko to prowadzi do wyższych kosztów operacyjnych.

Dodatkowe koszty są związane z przechowywaniem ogromnych ilości danych używanych do trenowania lub dostrajania i zarządzaniem nimi, a także z regularną konserwacją, taką jak aktualizowanie modeli, wdrażanie środków bezpieczeństwa i naprawianie błędów. Wszystko to sprawia, że koszty związane ze stosowaniem modeli LLM mogą być wyzwaniem. Jak w przypadku każdej technologii wykorzystywanej do celów biznesowych, ważnym zagadnieniem jest unikanie potencjalnych sporów prawnych i zapewnienie zgodności z przepisami. Kolejnym czynnikiem zwiększającym te koszty jest inwestowanie w ciągłe badania i rozwój w celu ulepszenia modeli i uzyskania przewagi konkurencyjnej.

Wspomnieliśmy wcześniej o technicznych problemach związanych z limitami tokenów, które prawdopodobnie zostaną rozwiązane, ale nie wspomnieliśmy w tym kontekście o ograniczaniu kosztów, a zagadnienia te są powiązane, gdyż większość interfejsów API nalicza opłaty na podstawie liczby tokenów. Oznacza to, że wysyłanie większej ilości danych kontekstowych i stosowanie lepszych promptów jest droższe. Utrudnia to również przewidywanie kosztów, ponieważ choć można standaryzować dane wejściowe, to standaryzowanie wyników nie jest możliwe. Nigdy nie będziemy mieli pewności, ile tokenów zostanie zwróconych, a to utrudnia kontrolowanie wydatków. Pamiętaj, że w przypadku modeli LLM jak nigdy wcześniej ważne jest wdrażanie i przestrzeganie odpowiednich praktyk inżynierii kosztów, aby zapewnić, że koszty nigdy nie wymkną się spod kontroli.

3.3. Podstawy LLMOps

Teraz gdy już masz w miarę całościowe pojęcie o wyzwaniu, z którym się mierzymy, przyjrzyjmy się różnym praktykom, narzędziom i infrastrukturze LLMOps, aby zobaczyć, jak poszczególne komponenty mogą nam pomóc pokonać te przeszkody. Najpierw przyjrzymy się różnym praktykom, począwszy od kompresji, w ramach której opisane zostaną metody zmniejszania, przycinania i aproksymacji, pozwalające maksymalnie redukować rozmiary modeli. Następnie

zajmiemy się przetwarzaniem rozproszonym, które jest niezbędne do uruchomienia modeli LLM, ponieważ są one tak duże, że rzadko mieszczą się w pamięci pojedynczego GPU. Po omówieniu tych zagadnień, w kolejnym punkcie podrozdziału, przeanalizujemy zagadnienia związane z infrastrukturą i narzędziami potrzebnymi do realizacji całego procesu.

3.3.1. Kompresja

Czytając o wyzwaniach związanych z dużymi modelami językowymi (LLM) przedstawionych w poprzednim rozdziale, mogłeś zadać sobie pytanie: „Skoro największe problemy LLM wynikają z ich rozmiaru, dlaczego po prostu ich nie zmniejszyć?”. Jeśli tak pomyślałeś, gratulacje! Jesteś geniuszem — właśnie do tego służy kompresja. Zmniejszenie modeli do najmniejszych możliwych rozmiarów poprawi czas wdrożenia, skróci opóźnienia, ograniczy liczbę potrzebnych drogich GPU i ostatecznie zaoszczędzi pieniądze. Jednak cały sens tworzenia tak oszałamiająco gigantycznych modeli polegał przede wszystkim na tym, że stawały się one lepsze w tym, co robią. Musimy umieć je zmniejszać bez utraty zalet, które modele te zapewniają ze względu na swoje ogromne rozmiary.

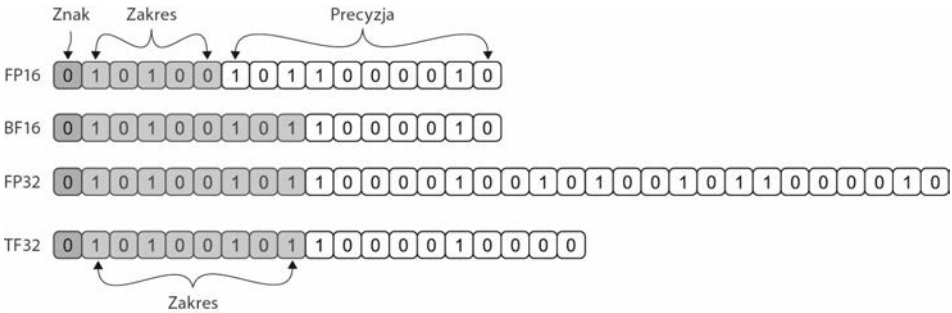
Do rozwiązania tego problemu wciąż daleka droga, ale istnieje wiele sposobów, by go ominąć, a każdy z nich ma swoje zalety i wady. W kolejnych podpunktach opisanych zostało kilka z tych metod, począwszy od najłatwiejszej i najbardziej efektywnej.

KWANTYZACJA

Kwantyzacja to proces zmniejszania precyzji w celu obniżenia wymagań pamięciowych. Już intuicja podpowiada, że ta wymiana ma sens. Kiedy autor był na studiach, uczono go, aby zawsze zaokrąglać liczby do precyzji narzędzi pomiarowych. Gdybym mierzył ołówek linijką, nikt by nie uwierzył w wynik 19,025467821973739 cm. Nawet z użyciem suwmiarki nie mógłby zweryfikować liczby z taką dokładnością. W razie zastosowania linijki każda liczba zapisana po 19,03 cm będzie fikcją. Aby to zobrazować: jeden z moich profesorów inżynierii zapytał kiedyś: „Czy mierząc wysokość wieżowca, przejmujesz się dodatkową kartką papieru na jego dachu?”.

Sposób reprezentowania liczb w komputerach często prowadzi nas do przekonania, że mamy lepszą precyzję niż w rzeczywistości. Aby to zilustrować, otwórz terminal Pythona i dodaj 0,1 + 0,2. Jeśli nigdy wcześniej tego nie próbowałeś, możesz być zaskoczony, że wynik to nie 0,3, ale 0,30000000000000004. Nie będziemy zagłębiać się w szczegóły matematyczne tego zjawiska, ale pytanie pozostaje: Czy możemy zmniejszyć precyzję bez pogorszenia wyników? Tak naprawdę potrzebujemy precyzji tylko do dziesiątego miejsca po przecinku, ale zmniejszenie precyzji prawdopodobnie da nam liczbę taką jak 0,304 zamiast 0,300, a tym samym zwiększy margines błędu.

Ostatecznie jedyne liczby, które komputer rozumie, to 0 i 1, coś, co jest włączone lub nie, pojedynczy bit. Aby poprawić ten zakres, łączymy wiele bitów i przypisujemy im różne znaczenia. Połącz razem 8 bitów, a otrzymasz bajt. Z użyciem standardu INT8 możemy zakodować w tym bajcie wszystkie liczby całkowite od –128 do 127. Oszczędzimy Ci szczegółów, zakładając, że już wiesz, jak działa system binarny; wystarczy powiedzieć, że im więcej mamy bitów, tym większy zakres liczb możemy reprezentować — zarówno liczb większych, jak i mniejszych. Na rysunku 3.1 pokazanych jest kilka popularnych sposobów zapisu liczb zmiennoprzecinkowych. W wyniku połączenia 32 bitów otrzymujemy to, co pretensjonalnie nazywamy *pełną precyzją* (ang. *full precision*), i tak przechowywana jest większość liczb, w tym wagi w modelach uczenia maszynowego. Podstawowa kwantyzacja przenosi nas z pełnej precyzji do połowy precyzji, zmniejszając modele o połowę. Istnieją dwa różne standardy połowy precyzji, FP16 i BF16, które różnią się liczbą bitów reprezentujących zakres i wykładnik. Ponieważ standard BF16 używa tej samej liczby bitów wykładnika, co FP32, okazał się bardziej efektywny w kwantyzacji i można generalnie oczekiwać prawie dokładnie tego samego poziomu dokładności przy połowie rozmiaru modelu. Jeśli zrozumiałeś analogię z kartką papieru i wieżowcem, powinno być dla Ciebie oczywiste, dlaczego tak jest.



Rysunek 3.1. *Odwzorowanie bitów w popularnych sposobach reprezentowania liczb zmiennoprzecinkowych: 16-bitowa liczba zmiennoprzecinkowa, nazywana także liczbą o połowie precyzji (FP16), 32-bitowa liczba zmiennoprzecinkowa, inaczej liczba o pełnej pojedynczej precyzji (FP32), oraz reprezentacja TensorFloat (TF32), stosowana przez firmę NVIDIA*

Jednak nie ma powodu, by na tym poprzestać. Często bez zbyt dużej utraty dokładności możemy zejść o kolejny bajt, do formatów 8-bitowych. Przeprowadzono już nawet udane badania pokazujące, że możliwa jest selektywna 4-bitowa kwantyzacja części modelu LLM, która wiąże się jedynie z niewielką utratą dokładności. Selektywne stosowanie kwantyzacji to proces znany jako kwantyzacja dynamiczna (ang. *dynamic quantization*), który zwykle dotyczy tylko wag — w wartościach aktywacji wciąż stosuje się pełną precyzję, co ma na celu minimalizację utraty dokładności.

Świętym Graalem kwantyzacji jest jednak INT2, reprezentujący każdą liczbę jako -1, 0 lub 1. Obecnie stosowanie tego zapisu bez całkowitej degradacji modelu nie jest możliwe, ale pozwoliłby on zmniejszyć rozmiary modeli nawet ośmiokrotnie. Model Bloom miałby zaledwie około 40 GB, co wystarczyłoby, aby zmieścił się w pamięci pojedynczego GPU. To oczywiście maksimum tego, co można osiągnąć za pomocą kwantyzacji. Aby jeszcze bardziej zmniejszyć model, musielibyśmy sięgnąć po dodatkowe metody.

Najlepsze w kwantyzacji jest to, że łatwo ją przeprowadzić. Istnieje wiele frameworków, które na to pozwalają. Przykład z listingu 3.1 pokazuje zaś, w jaki sposób można użyć biblioteki kwantyzacji PyTorch, by po zakończeniu trenowania modelu wykonać prostą kwantyzację statyczną (ang. *post-training static quantization*, PTQ). Wszystko, czego potrzebujemy, to model korzystający z pełnej precyzji, kilka przykładowych danych wejściowych i zbiór walidacyjny do przygotowania i kalibracji. Jak widać, wszystko sprowadza się jedynie do kilku wierszy kodu.

Listing 3.1. Przykład kwantyzacji po wytrenowaniu modelu w bibliotece PyTorch

```
import copy
import torch
import torch.ao.quantization as q

model_to_quantize = copy.deepcopy(model_fp32)
model_to_quantize.eval()

model_to_quantize.qconfig = q.get_default_qconfig("qnnpack")

prepared_model = q.prepare(model_to_quantize)

with torch.inference_mode():
    for x in dataset:
        prepared_model(x)

model_quantized = q.convert(prepared_model)
```

Wykonujemy głęboką kopię oryginalnego modelu, ponieważ kwantyzacja odbywa się „w miejscu”

Pobieramy konfigurację — używamy „qnnpack” dla procesorów ARM i „fbgemm” dla procesorów x86

Przygotowanie

Kalibracja — należy użyć reprezentatywnych danych (walidacyjnych)

Kwantyzacja

Statyczna kwantyzacja PTQ to najprostsze podejście do kwantyzacji. Wykonuje się ją po zakończeniu treningu modelu, a kwantyzacji podlegają równomiernie wszystkie parametry. Jak w przypadku większości metod, najprostsze podejście wprowadza więcej błędów. Często są one do przyjęcia, ale gdy tak nie jest, możemy wprowadzić dodatkową złożoność, aby ograniczyć utratę dokładności wynikającą z kwantyzacji. Warto rozważyć takie metody jak kwantyzacja jednorodna lub niejednorodna, statyczna lub dynamiczna, symetryczna lub asymetryczna oraz stosowana podczas trenowania lub po jego zakończeniu.

Aby zrozumieć te metody, rozważmy przypadek, w którym przeprowadzamy kwantyzację z formatu FP32 do INT8. W formacie FP32 mamy do dyspozycji praktycznie pełny zakres liczb, ale w INT8 tylko 256 wartości. Próbujemy zatem zamknąć dżina w butelce, a nie jest to łatwe zadanie. Jeśli przyjrzysz się

wagom swojego modelu, możesz zauważyć, że większość liczb to ułamki z przedziału [–1, 1]. Moglibyśmy to wykorzystać przez zastosowanie standardu 8-bitowego, który w sposób niejednorodny reprezentuje więcej wartości z tego zakresu, zamiast standardowego jednorodnego zakresu [–128, 127]. Choć jest to matematycznie możliwe, to jednak takie standardy nie są powszechne, a współczesnego sprzętu i oprogramowania do głębokiego uczenia nie projektowano z myślą o ich wykorzystaniu. Dlatego na razie najlepiej trzymać się kwantyzacji jednorodnej.

Najprostszym podejściem do zmniejszenia danych jest ich normalizacja, ale ponieważ przechodzimy ze skali ciągłej na dyskretną, pojawia się kilka pułapek, które warto omówić. Zaczynamy od znalezienia wartości minimalnej i maksymalnej, a następnie skalujemy je do nowego zakresu liczbowego. Następnie przypisujemy wszystkie pozostałe liczby do odpowiednich przedziałów. Oczywiście, jeśli mamy bardzo duże wartości odstające, może się okazać, że wszystkie pozostałe liczby zostaną przypisane do jednego lub dwóch przedziałów, a w rezultacie utracą wszelką pierwotną ziarnistość. Aby temu zapobiec, możemy obciąć duże liczby — to właśnie robimy w przypadku kwantyzacji statycznej. Jednak zanim obetniemy dane, warto zastanowić się, co by się stało, gdybyśmy wcześniej wybrali zakres i skalę, które obejmują większość naszych danych. Musimy być ostrożni, ponieważ jeśli ten dynamiczny zakres będzie zbyt mały, wprowadzimy więcej błędów przycinania; a jeśli będzie zbyt duży, wprowadzimy więcej błędów zaokrąglania. Celem kwantyzacji dynamicznej jest oczywiście zmniejszenie obu tych rodzajów błędów.

Następnie musimy rozważyć symetrię danych. Zazwyczaj w normalizacji wymuszamy, aby dane miały rozkład normalny, a więc symetryczny. Możemy jednak zdecydować się na skalowanie danych w sposób, który zachowa ich asymetrię. Dzięki temu potencjalnie moglibyśmy zmniejszyć ogólną stratę spowodowaną błędami obcięcia i zaokrąglenia, choć nie ma takich gwarancji.

W ostateczności, jeśli żadna z tych metod nie zmniejszy utraty dokładności modelu, możemy zastosować trening uwzględniający kwantyzację (ang. *quantization-aware training*, QAT). Jest to prosty proces, w którym podczas trenowania modelu uwzględniamy dodatkowy krok symulowanej kwantyzacji. W tym przypadku „symulowana” oznacza, że obcinamy i zaokrąglamy dane, zachowując ich pełną precyzję. Pozwala to modelowi już podczas treningu dostosować się do błędów i odchyleń wprowadzanych przez kwantyzację. QAT zapewnia wyższą dokładność niż inne metody, ale za cenę znacznie dłuższego czasu trenowania modelu.

Metody kwantyzacji

- *Reprezentacja jednolita czy nierównomierna* — Czy stosować 8-bitowy standard o jednolitym zakresie wartości, czy też nierównomierny, aby uzyskać większą precyzję w przedziale wartości od –1 do 1.
- *Statyczna czy dynamiczna* — Wybór dostosowania zakresu lub skali przed przycięciem w celu zmniejszenia błędów przycinania i zaokrąglania oraz ograniczenia utraty danych.
- *Symetryczna czy asymetryczna* — Normalizacja danych do rozkładu normalnego i wymuszenie symetrii lub zachowanie asymetrii i skośności.
- *Podczas trenowania czy po nim* — Kwantyzacja po treningu jest bardzo prosta do wykonania, a chociaż przeprowadzenie jej podczas treningu wymaga więcej pracy, prowadzi to do zmniejszenia błędów systematycznych i lepszych wyników.

Kwantyzacja to niezwykle potężne narzędzie. Zmniejsza rozmiar modelu i obciążenie obliczeniowe potrzebne do jego uruchomienia, co przekłada się na mniejsze opóźnienia i koszty działania. Najlepsze w kwantyzacji jest to, że można ją wykonać po fakcie, więc nie trzeba się martwić, czy naukowcy zajmujący się danymi pamiętali o przeprowadzeniu kwantyzacji modelu podczas jego trenowania z użyciem takich metod jak QAT. Dlatego kwantyzacja stała się tak popularna w pracy z dużymi modelami językowymi (LLM) i innymi rozbudowanymi modelami uczenia maszynowego. Chociaż w trakcie stosowania technik kompresji zawsze pojawiają się obawy przed zmniejszeniem dokładności, to jednak w porównaniu z innymi metodami kwantyzacja jest rozwiązaniem korzystnym pod każdym względem.

PRZYCINANIE

Gratulacje, właśnie wytrenowałeś zupełnie nowy model LLM! Skoro ma miliardy parametrów, to wszystkie muszą być użyteczne, prawda? Niestety nie! Jak to bywa z większością rzeczy w życiu, także parametry modelu zwykle podlegają zasadzie Pareto. Około 20% wag decyduje o 80% wartości. Możesz zadać sobie pytanie: „Jeśli to prawda, to dlaczego po prostu nie usuniemy całego zbędnego balastu?" Świetny pomysł! Możesz poklepać się po plecach. Przycinanie (ang. *pruning*) to proces wycinania i usuwania tych części modelu, które uznajemy za bezwartościowe.

Istnieją zasadniczo dwie metody przycinania: *strukturalne* i *niestrukturalne*. Przycinanie strukturalne polega na znajdowaniu elementów strukturalnych modelu, które nie przyczyniają się do jego wydajności, a następnie ich usuwaniu — czy to filtrów, kanałów, czy warstw w sieci neuronowej. Zaletą tej metody jest to, że model staje się nieco mniejszy, ale zachowuje tę samą podstawową strukturę, co oznacza, że nie musimy obawiać się utraty efektywności sprzętowej. Gwarantuje to również zmniejszenie opóźnień, ponieważ model będzie miał mniej obliczeń do wykonania.

Z drugiej strony przycinanie niestrukturalne polega na przeglądaniu parametrów i zerowaniu tych mniej istotnych, które niewiele wnoszą do wydajności modelu. W przeciwieństwie do przycinania strukturalnego, w tym przypadku nie usuwamy żadnych parametrów; po prostu ustawiamy je na zero. Możemy sobie wyobrazić, że dobrym punktem wyjścia byłyby wagi lub aktywacje już bliskie 0. Oczywiście, chociaż skutecznie zmniejsza to rozmiar modelu, to zarazem oznacza, że nie eliminujemy żadnych obliczeń, więc często obserwuje się tylko minimalną, o ile w ogóle jakąkolwiek, poprawę pod względem opóźnień. Ale mniejszy model nadal oznacza krótsze czasy ładowania i mniej GPU potrzebnych do jego uruchomienia. Zapewnia nam to również bardzo precyzyjną kontrolę nad procesem, pozwalając na większe zmniejszenie modelu niż w przypadku przycinania strukturalnego przy jednoczesnym mniejszym wpływie na wydajność.

Podobnie jak kwantyzację przycinanie można przeprowadzić po wytrenowaniu modelu. Jednak w przeciwieństwie do kwantyzacji powszechną praktyką jest dodatkowe dostrajanie, aby zapobiec zbyt dużej utracie wydajności. Coraz częściej podczas trenowania modelu stosuje się włączanie kroków przycinania, aby uniknąć konieczności późniejszego dostrajania. Ponieważ bardziej rzadki model (ang. *sparse model*) będzie miał mniej parametrów do dostrojenia, dodanie tych kroków przycinania może również pomóc modelowi szybciej osiągnąć zbieżność[5].

Będziesz zaskoczony, jak bardzo można zmniejszyć model z zastosowaniem przycinania przy jednoczesnym minimalnym wpływie na jego wydajność. Jak bardzo? W artykule o SparseGPT[6] omówiono metodę, która próbuje automatycznie przeprowadzić proces przycinania w jednym kroku, bez konieczności późniejszego dostrajania. Autorzy odkryli, że bez problemu mogą zmniejszyć model GPT-3 o 50 – 60%! W zależności od modelu i zadania w niektórych przypadkach zaobserwowali nawet niewielką poprawę wydajności. Z niecierpliwością czekamy, żeby się przekonać, dokąd zaprowadzi nas przycinanie w przyszłości.

DESTYLACJA WIEDZY

Destylacja wiedzy to prawdopodobnie najciekawsza metoda kompresji modeli językowych. Idea jest prosta: wykorzystujemy duży model językowy do wytrenowania mniejszego modelu, który będzie go naśladował. Zaletą tej metody jest to, że większy model zapewnia praktycznie nieskończony zbiór danych treningowych dla mniejszego modelu, co może sprawić, że trening będzie bardzo efektywny. Ponieważ im większy zbiór danych, tym lepsza wydajność, często

[5] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden i A. Peste, *Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks*, styczeń 2021 r., *https://arxiv.org/abs/2102.00554*.

[6] E. Frantar i D. Alistarh, *SparseGPT: Massive Language models can be accurately pruned in one-shot*, styczeń 2023 r., *https://arxiv.org/abs/2301.00774*.

obserwujemy, że mniejsze modele osiągają niemal taki sam poziom dokładności jak te użyte do ich uczenia[7].

Model wytrenowany w ten sposób z pewnością będzie mniejszy i szybszy. Wadą tej metody jest konieczność wytrenowania zupełnie nowego modelu, co wiąże się ze znacznymi kosztami początkowymi. Wszelkie przyszłe ulepszenia modelu nauczyciela będą musiały zostać przekazane modelowi ucznia, co może prowadzić do skomplikowanych cykli treningowych i złożonej struktury wersji. Jak widać, metoda ta wymaga zdecydowanie większych nakładów pracy niż niektóre inne metody kompresji.

W metodzie destylacji wiedzy najtrudniejsze jest to, że wciąż nie mamy dobrych, sprawdzonych przepisów na jej stosowanie. Trudne pytania, takie jak „Jak mały może być model ucznia?", dopiero trzeba będzie rozwiązywać metodą prób i błędów. Jest jeszcze wiele do nauczenia się i zbadania w tej dziedzinie.

Jednak pojawiły się już pewne obiecujące prace z tego zakresu, na przykład projekt Alpaca8, rozwijany na Uniwersytecie Stanforda. Zamiast trenować model ucznia od podstaw, twórcy tego projektu zdecydowali się na dostrojenie otwartoźródłowego modelu Llama o 7 miliardach parametrów, używając jako nauczyciela modelu GPT-3.5 firmy OpenAI o 175 miliardach parametrów. Sama idea była prosta, ale w ocenie badaczy przyniosła świetne rezultaty. Największym zaskoczeniem był koszt — twórcy wydali tylko 500 dolarów na pokrycie kosztów użycia API, niezbędnego do uzyskania danych treningowych od modelu nauczyciela, i 100 dolarów na czas GPU, potrzebny do dostrojenia modelu ucznia. Oczywiście, gdyby ktoś chciał wykorzystać to w zastosowaniu komercyjnym, naruszyłby warunki korzystania z usług OpenAI, więc jako nauczycieli najlepiej będzie używać modeli własnych lub otwartoźródłowych.

APROKSYMACJA MACIERZY PRZEZ MACIERZE NISKIEGO RZĘDU

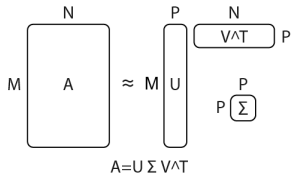
Aproksymacja macierzy przez macierze niskiego rzędu (ang. *low-rank approximation*), znana również jako faktoryzacja niskorzędowa, rozkład niskiego rzędu lub faktoryzacja macierzy (to wszystko przez matematyków, którzy nadali jej zbyt wiele nazw), wykorzystuje sztuczki z algebry liniowej do uproszczenia dużych macierzy lub tensorów w celu znalezienia ich reprezentacji o niższym wymiarze. Istnieje kilka technik, które na to pozwalają. Najczęściej stosowane to rozkład według wartości osobliwych (SVD), rozkład Tuckera (TD) oraz kanoniczna dekompozycja poliadyczna (ang. *canonical polyadic decomposition*, CPD).

Na rysunku 3.2 przedstawiliśmy ogólną ideę działania metody SVD. W zasadzie rozbijamy bardzo dużą macierz A na trzy mniejsze macierze: U, Σ i V.

[7] V. Sanh, L. Debut, J. Chaumond i T. Wolf, *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*, październik 2019 r., *https://arxiv.org/abs/1910.01108*.

[8] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P Liang i T.B. Hashimoto, *Alpaca: A strong, replicable instruction-following model*, CRFM, 2023 r., *https://crfm.stanford.edu/2023/03/13/alpaca.html*.

Macierze U i V służą do zachowania wymiarów i względnych wartości oryginalnej macierzy, a trzecia z nich, Σ, pozwala nam zastosować kierunek i przesunięcie. Im mniejsza jest macierz Σ, tym bardziej redukujemy całkowitą liczbę parametrów, ale jednocześnie zmniejszamy dokładność aproksymacji.



Rysunek 3.2. *Przykład SVD — aproksymacji macierzy przez macierze niskiego rzędu. A to duża macierz o wymiarach N i M. Możemy ją przybliżyć trzema mniejszymi: macierzą U o wymiarach M i P, macierzą Σ, będącą macierzą kwadratową o wymiarze P, oraz macierzą V o wymiarach N i P (tutaj pokazujemy jej transpozycję). Zazwyczaj spełnione są warunki P<<M oraz P<<N*

Aby lepiej zrozumieć tę koncepcję, warto zobaczyć konkretny przykład. W listingu 3.2 pokazujemy prosty przykład działania metody SVD przy kompresji macierzy 4×4. Do tego potrzebujemy tylko podstawowych bibliotek SciPy i NumPy, które importujemy w wierszach 1. i 2. W wierszu 3. definiujemy macierz, a następnie w wierszu 9. stosujemy do niej rozkład SVD.

Listing 3.2. Przykład aproksymacji macierzami niskiego rzędu z użyciem SVD

```
import scipy
import numpy as np
matrix = np.array([
    [ 1., 2., 3., 4.],
    [ 5., 6., 7., 8.],
    [ 9., 10., 11., 12.],
    [13., 14., 15., 16.]
])
u, s, vt = scipy.sparse.linalg.svds(matrix, k=1)
print(u,s,vt)
# [[-0.13472211]
# [-0.34075767]
# [-0.5467932 ]
```

Wygenerowane wyniki to

```
[-0.7528288 ]], [38.62266], [[-0.4284123 -0.47437257 -0.52033263 –0.5662928 ]]
```

Przyglądając się bliżej macierzom U, Σ i transpozycji V, widzimy że mają one wymiary, odpowiednio, 4×1, 1×1 i 1×4. W rezultacie potrzebujemy teraz tylko 9 parametrów zamiast pierwotnych 16, co zmniejsza zużycie pamięci niemal o połowę.

Na koniec mnożymy te macierze z powrotem, aby uzyskać przybliżenie pierwotnej macierzy. W tym przypadku aproksymacja nie jest zbyt dokładna, ale wciąż można zauważyć, że ogólny porządek i wielkości odpowiadają pierwotnej macierzy:

```
svd_matrix = u*s*vt
print(svd_matrix)
```

Wygenerowane wyniki to

```
array([[ 2.2291691, 2.4683154, 2.7074606, 2.9466066],
       [ 5.6383204, 6.243202 , 6.848081 , 7.4529614],
       [ 9.047472 , 10.018089 , 10.988702 , 11.959317 ],
       [12.456624 , 13.792976 , 15.129323 , 16.465673 ]], dtype=float32)
```

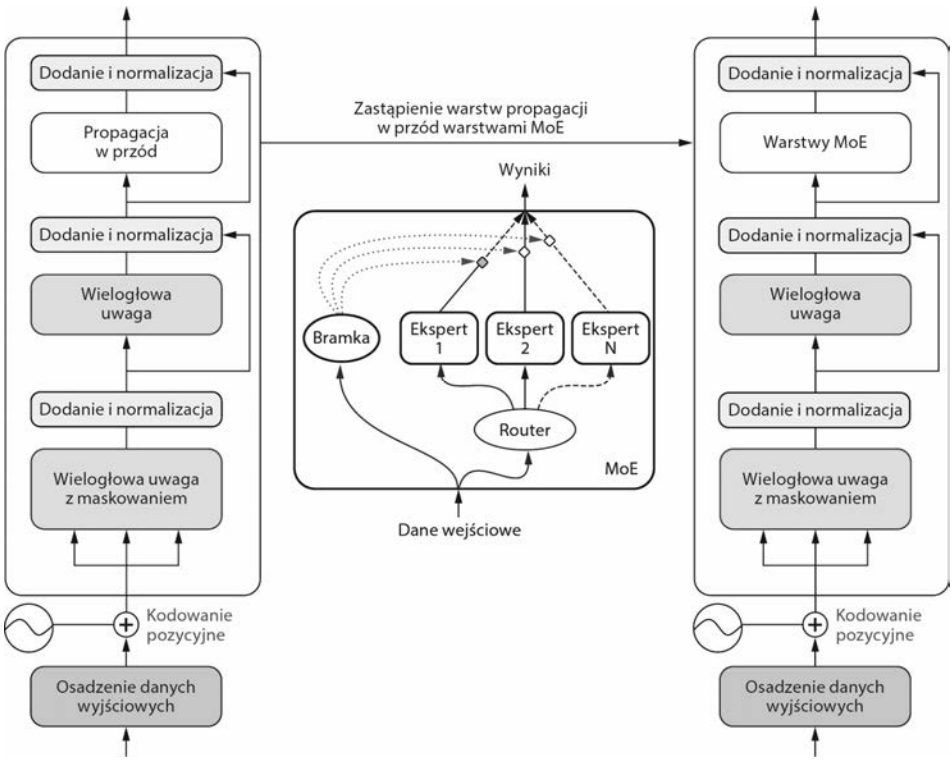
Niestety, nie znamy przypadków wykorzystania tej metody do kompresji modeli w środowisku produkcyjnym, najprawdopodobniej ze względu na niską dokładność aproksymacji. Jednak stosuje się ją — i to jest istotne — do adaptacji i dostrajania modeli, co wiąże się z techniką adaptacji niskorzędowej (LoRA)[9]. Adaptacja to proces dostrajania modelu ogólnego lub bazowego do wykonywania konkretnego zadania. Technika LoRA wykorzystuje aproksymację niskiego rzędu SVD do wag uwagi, a dokładniej: do wprowadzenia macierzy aktualizacji działających równolegle do wag uwagi, co pozwala na dostrajanie znacznie mniejszego modelu. Technika LoRA zyskała dużą popularność, ponieważ umożliwia łatwe zmniejszenie podlegających trenowaniu warstw dużego modelu językowego do niewielkiego ułamka oryginalnego rozmiaru oraz na późniejsze trenowanie takiego modelu na standardowym sprzęcie. Eksperymenty ze stosowaniem tej techniki można zacząć od wykorzystania biblioteki PEFT udostępnionej przez Hugging Face, gdzie można także znaleźć kilka poradników na ten temat.

UWAGA Dla bardziej dociekliwych: dostrajanie efektywne parametrowo (PEFT) to klasa metod mających na celu dostrajanie modeli w sposób wydajny obliczeniowo. Biblioteka PEFT ma na celu zgromadzenie ich wszystkich w jednym, łatwo dostępnym miejscu. Więcej informacji na jej temat można znaleźć na stronie *https://huggingface.co/docs/peft*.

MIESZANINA EKSPERTÓW

Mieszanina ekspertów (ang. *mixture of experts*, MoE) to technika, w której zastępujemy warstwy propagacji w przód (ang. *feed-forward*) w transformerze warstwami MoE. Warstwy propagacji w przód słyną z dużej liczby parametrów i intensywnych obliczeń, więc zastąpienie ich czymś lepszym może mieć duży wpływ na wydajność. MoE to grupa modeli aktywowanych rzadko. Różnią się one od technik zespołowych tym, że zamiast łączenia wyników ze wszystkich modeli zazwyczaj uruchamiany jest tylko jeden model ekspercki lub kilka. Rzadkość jest często wywoływana przez mechanizm bramkowania, który uczy się, których ekspertów używać, i/lub mechanizm routingu, który określa, których ekspertów w ogóle należy brać pod uwagę. Rysunek 3.3 przedstawia architekturę MoE używającą potencjalnie N ekspertów oraz pokazuje jej miejsce w stosie dekodera.

[9] E.J. Hu i inni, *LoRA: Low-rank adaptation of large language models*, czerwiec 2021 r., *https://arxiv.org/abs/2106.09685*.



Rysunek 3.3. *Przykład modelu mieszaniny ekspertów (MoE) z bramką i routerem kontrolującymi przepływ. Model MoE zastępuje warstwy FFN w transformerze; tutaj pokazano go zastępującego FFN w dekoderze*

W zależności od liczby ekspertów warstwa MoE może potencjalnie mieć więcej parametrów niż warstwy propagacji w przód (FFN), co prowadzi do większego modelu. W praktyce jednak tak się nie dzieje, ponieważ inżynierowie i badacze dążą do stworzenia mniejszego modelu. Na pewno zobaczymy szybszą ścieżkę obliczeń i krótszy czas wnioskowania. Jednak tym, co naprawdę wyróżnia technikę MoE, jest połączenie jej z kwantyzacją. Jedno z badań[10] przeprowadzonych wspólnie przez firmy Microsoft i NVIDIA wykazało, że dzięki wykorzystaniu techniki MoE udało się zastosować 2-bitową kwantyzację przy minimalnym wpływie na dokładność!

Oczywiście, ponieważ jest to dość duża zmiana w strukturze modelu, konieczne będzie późniejsze dostrojenie. Należy również pamiętać, że warstwy MoE często zmniejszają zdolność modelu do uogólniania, więc najlepiej sprawdzają się w modelach zaprojektowanych do konkretnych zadań. Istnieje kilka bibliotek implementujących warstwy MoE; my zachęcamy do wypróbowania biblioteki DeepSpeed.

[10] R. Henry i Y.J. Kim, *Accelerating large language models via low-bit quantization*, marzec 2023 r., *https://www.nvidia.com/en-us/on-demand/session/gtcspring23-s51226/*.

UWAGA DeepSpeed to biblioteka, która optymalizuje wiele trudnych aspektów związanych z modelami głębokiego uczenia na dużą skalę, takimi jak LLM, i jest szczególnie przydatna podczas ich trenowania. Samouczek pokazujący, jak stosować technikę MoE przy jej użyciu, można znaleźć na stronie *https://www.deepspeed.ai/tutorials/mixture-of-experts/*.

3.3.2. Przetwarzanie rozproszone

Przetwarzanie rozproszone to technika stosowana w uczeniu głębokim w celu zrównoleglenia i przyspieszenia działania dużych, złożonych sieci neuronowych przez podział obciążenia na wiele urządzeń lub węzłów w klastrze. Takie podejście znacząco skraca czas treningu i wnioskowania, umożliwiając równoczesne obliczenia oraz równoległy dostęp do danych i modelu. Wraz z ciągłym wzrostem wielkości zbiorów danych i złożoności modeli, przetwarzanie rozproszone stało się kluczowe dla procesów uczenia głębokiego: zapewniło efektywne wykorzystanie zasobów i umożliwiło badaczom skuteczne prowadzenie prac nad modelami. Przetwarzanie rozproszone jest jedną z podstawowych praktyk odróżniających uczenie głębokie od uczenia maszynowego, a w przypadku dużych modeli językowych musimy zastosować wszystkie dostępne techniki. Przyjrzyjmy się zatem różnym praktykom przetwarzania równoległego, aby móc w pełni wykorzystać możliwości przetwarzania rozproszonego.

RÓWNOLEGŁOŚĆ DANYCH

Równoległość danych (ang. *data parallelism*) to metoda, o której najczęściej się wspomina w kontekście równoległego uruchamiania procesów; jest też najłatwiejsza do wdrożenia. Polega na podzieleniu danych i przetwarzaniu ich przez wiele kopii modelu lub potoku. W większości frameworków taki równoległy dostęp do danych można skonfigurować bardzo łatwo; na przykład w PyTorch można w tym celu użyć metody `DistributedDataParallel`. Jednak w większości takich konfiguracji pojawia się jeden problem: model musi się zmieścić na jednym GPU. A o to możemy zadbać z użyciem przydatnych narzędzi, takich jak Ray.io.

Ray.io, lub po prostu Ray, to projekt open-source stworzony z myślą o przetwarzaniu rozproszonym, szczególnie ukierunkowany na obliczenia równoległe i klastrowe. To elastyczne i przyjazne dla użytkownika narzędzie, które upraszcza programowanie rozproszone i pomaga programistom łatwo wykonywać zadania współbieżnie. Ray jest przede wszystkim przeznaczone do wykorzystania w zagadnieniach uczenia maszynowego oraz w rozwiązaniach wymagających wysokiej wydajności; jednak można go także używać w innych zastosowaniach. Kod przedstawiony na listingu 3.3 to prosty przykład pokazujący, jak można użyć Ray do rozproszenia zadania. Piękno Ray wynika z prostoty — wszystko, co musimy zrobić, aby nasz kod działał równolegle, to dodać dekorator. To znacznie lepsze rozwiązanie niż złożone konfiguracje kodu wykonywanego wielowątkowo lub asynchronicznie.

Listing 3.3. Przykład zapewniania równoległości danych z użyciem narzędzia Ray

```
import ray
import time

ray.init()

def slow_function(x):
    time.sleep(1)
    return x

@ray.remote
def slow_function_ray(x):
    time.sleep(1)
    return x

results = [slow_function(i) for i in range(1, 11)]

results_future = [slow_function_ray.remote(i) for i in range(1, 11)]
results_ray = ray.get(results_future)

print("Wyniki bez stosowania biblioteki Ray: ", results)
print("Wyniki z użyciem biblioteki Ray: ", results_ray)

ray.shutdown()
```

Uruchamiamy Ray

Definicja zwyczajnej funkcji Pythona

Przekształcamy funkcję w zadanie biblioteki Ray

Wykonujemy wolną funkcję bez stosowania Ray (co zabiera około 10 sekund)

Wykonujemy wolną funkcję z użyciem Ray (co zabiera 1 sekundę)

W Ray do zarządzania przetwarzaniem rozproszonym wykorzystywane są koncepcje zadań i aktorów. Zadania to funkcje, a aktorzy to obiekty z pamięcią stanu, które mogą być wywoływane i uruchamiane współbieżnie. Gdy wykonujesz zadania za pomocą Ray, system rozprowadza je na dostępne zasoby (np. wielordzeniowe procesory CPU lub wiele węzłów w klastrze). W przypadku modeli LLM musielibyśmy skonfigurować klaster Ray w środowisku chmurowym, co pozwoliłoby na uruchomienie każdego potoku na węźle z odpowiednią liczbą GPU, a to z kolei znacznie uprościłoby infrastrukturę potrzebną do równoległego uruchamiania modeli LLM.

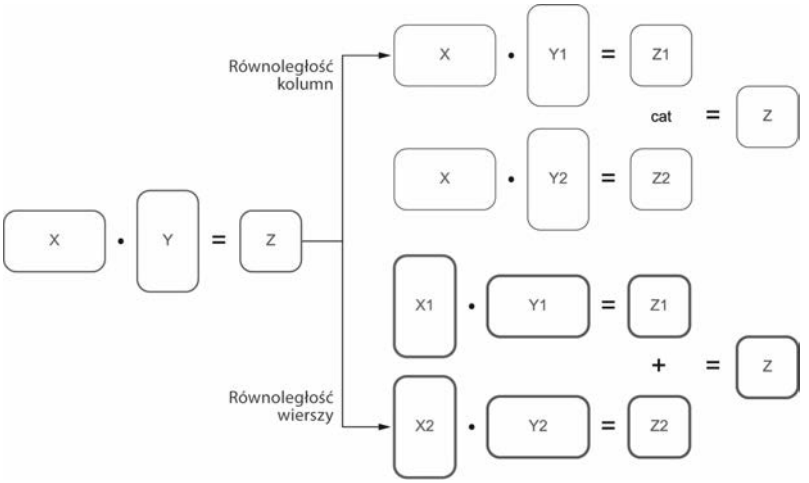
UWAGA Więcej informacji o klastrach Ray znajdziesz na stronie *https://docs.ray.io/en/releases-2.3.0/cluster/key-concepts.html#ray-cluster*.

Istnieje wiele alternatywnych rozwiązań, ale Ray zyskuje coraz większą popularność, ponieważ coraz więcej procesów uczenia maszynowego wymaga treningu rozproszonego. Wiele zespołów stosujących Ray uzyskało świetne efekty. Dzięki zastosowaniu Ray programiści mogą zapewnić lepszą wydajność i efektywniejsze wykorzystanie zasobów w rozproszonych przepływach pracy.

RÓWNOLEGŁOŚĆ TENSOROWA

Równoległość tensorowa wykorzystuje właściwości mnożenia macierzy do rozdzielenia aktywacji na wiele procesorów, przetworzenia danych, a następnie połączenia wyników po drugiej stronie procesorów. Na rysunku 3.4 pokazaliśmy, jak ten proces działa w przypadku macierzy, którą można przetwarzać równolegle

na dwa różne sposoby, dające ten sam wynik. Wyobraźmy sobie, że Y to bardzo duża macierz, która nie mieści się na pojedynczym procesorze lub, co bardziej prawdopodobne, stanowi wąskie gardło w naszym przepływie danych, przez co obliczenia trwają zbyt długo. W obu przypadkach możemy podzielić macierz Y na kolumny lub wiersze, wykonać obliczenia, a następnie połączyć wyniki. W tym przykładzie mamy do czynienia z macierzami, ale w rzeczywistości często posługujemy się tensorami mających więcej niż dwa wymiary. Jednak nadal mają zastosowanie te same zasady matematyczne, które sprawiają, że to działa.

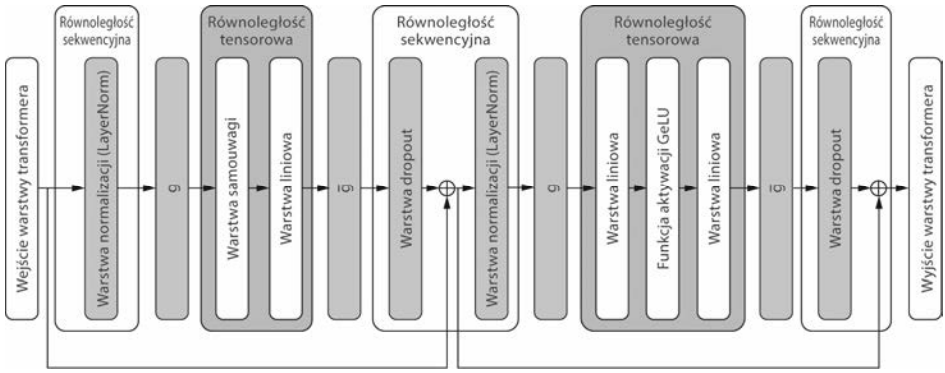


Rysunek 3.4. *Przykład równoległości tensorowej pokazujący, że można podzielić tensory według różnych wymiarów i uzyskać ten sam końcowy wynik. Porównujemy tu równoległość kolumn i wierszy macierzy*

Wybór wymiaru do zrównoleglenia jest w pewnym stopniu sztuką, ale jest kilka rzeczy, o których warto pamiętać, aby ułatwić sobie tę decyzję. Po pierwsze, ile mamy kolumn lub wierszy? Ogólnie rzecz biorąc, chcemy wybrać wymiar, który ma więcej elementów, niż wynosi liczba dostępnych procesorów, w przeciwnym razie przetwarzanie zakończy się przedwcześnie. Zazwyczaj nie będzie to stanowić problemu, ale dzięki użyciu narzędzi takich jak Ray, o których wspomnieliśmy w poprzednim podpunkcie, wykonywanie obliczeń równoległych w klastrze i uruchamianie wielu procesów jest bardzo proste. Po drugie, różne wymiary mają różne koszty powielania. Na przykład równoległość kolumn wymaga wysłania do każdego procesu całego zbioru danych, ma natomiast tę zaletę, że łączenie wyników jest szybkie i łatwe. Z kolei równoległość wierszy pozwala nam podzielić zbiór danych na części, ale wymaga dodawania wyników, co jest operacją bardziej kosztowną niż łączenie. Jak widać, pierwsza z tych opcji jest bardziej ograniczona przez wejście-wyjście, podczas gdy druga przez obliczenia. Ostatecznie najlepszy wymiar będzie zależał od konkretnego zbioru danych i ograniczeń sprzętowych. Pełna optymalizacja wymaga eksperymentowania, ale dobrym domyślnym wyborem jest po prostu największy wymiar.

Równoległość tensorowa pozwala nam rozdzielić warstwy intensywne obliczeniowo, takie jak warstwy perceptronów wielowarstwowych (MLP) i warstwy uwagi, na różne urządzenia, ale nie pomaga w przypadku warstw normalizacji czy dropout, które nie wykorzystują tensorów. Aby uzyskać lepszą ogólną wydajność potoku, możemy dodać równoległość sekwencyjną, która koncentruje się na tych blokach[11]. Równoległość sekwencyjna to proces, który dzieli aktywacje wzdłuż wymiaru sekwencji, co zapobiega nadmiarowemu przechowywaniu danych, i może być łączony z równoległością tensorową, co pozwala na osiągnięcie znacznych oszczędności pamięci przy minimalnym dodatkowym narzucie obliczeniowym. Połączenie tych dwóch rozwiązań pozwala zredukować ilość pamięci potrzebnej do przechowywania aktywacji w modelach transformerów. W rzeczywistości prawie eliminują one konieczność ponownego obliczania aktywacji i oszczędzają do pięciu razy więcej pamięci na aktywacje.

Rysunek 3.5 pokazuje, jak połączenie równoległości tensorowej, która pozwala nam rozdzielić intensywne obliczeniowo warstwy, z równoległością sekwencyjną, która tak samo rozdziela warstwy ograniczające pamięć, umożliwia zapewnienie równoległego przetwarzania całego modelu transformera. Razem pozwalają one na niezwykle efektywne wykorzystanie zasobów.



Rysunek 3.5. Połączenie równoległości tensorowej, koncentrujące się na warstwach intensywnych obliczeniowo, z równoległością sekwencyjną, redukującą zużycie pamięci, tworzy w pełni równoległy proces przetwarzania całego transformera

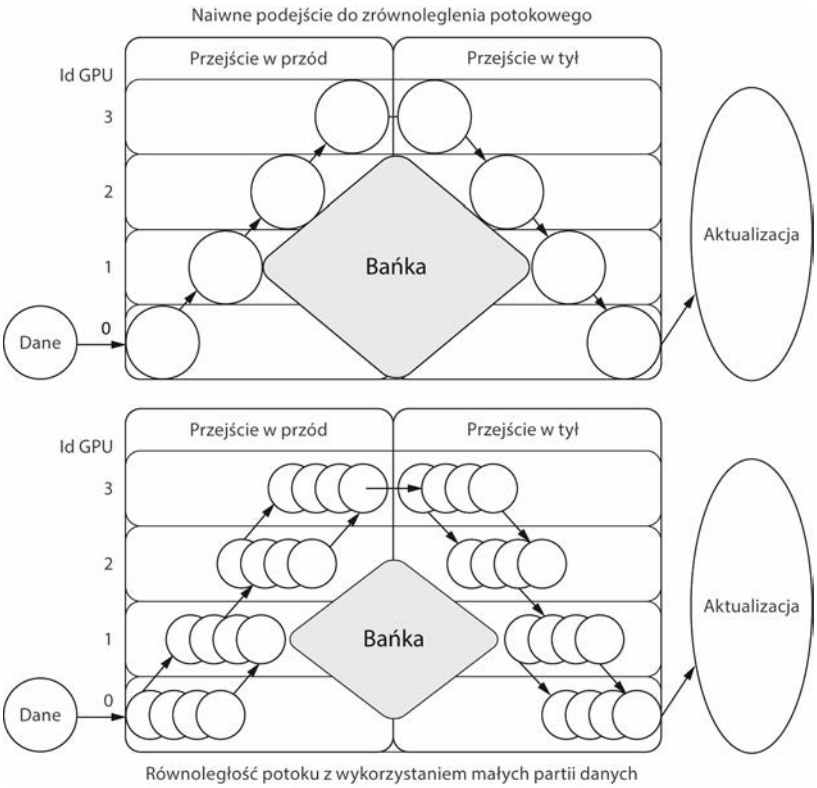
ZRÓWNOLEGLANIE POTOKOWE

Do tej pory mogliśmy przetwarzać duże ilości danych i przyspieszać wąskie gardła, ale to wszystko nie ma znaczenia, jeśli nasz model jest zbyt duży, by zmieścić się w pamięci pojedynczego GPU. Tu właśnie przydaje się zrównoleglenie potokowe — polega ono na podziale modelu w pionie i umieszczeniu każdej jego części na innym GPU. W efekcie powstaje potok, w którym dane wejściowe trafiają do pierwszego GPU, są na nim przetwarzane, a następnie są przesyłane do kolejnego

[11] V. Korthikanti i inni, *Reducing activation recomputation in large transformer models*, maj 2022 r., *https://arxiv.org/abs/2205.05198*.

GPU i tak dalej, aż przejdą przez cały model. Podczas gdy inne techniki zrównoleglania poprawiają moc obliczeniową i przyspieszają wnioskowanie, zrównoleglenie potokowe jest niezbędne, aby w ogóle można było uruchomić model. Ma ono jednak poważne wady, głównie związane z wykorzystaniem urządzeń.

Aby zrozumieć, skąd bierze się ta wada i jak minimalizować jej wpływ, rozważmy najpierw naiwne podejście, w którym po prostu używamy modelu, by jednocześnie przetworzyć wszystkie dane. Okazuje się, że takie podejście pozostawia ogromną „bańkę" niewykorzystanych zasobów. Ponieważ model jest podzielony, musimy sekwencyjnie przetwarzać na poszczególnych urządzeniach. Oznacza to, że gdy jeden GPU przetwarza dane, pozostałe są bezczynne. Na rysunku 3.6 pokazaliśmy to naiwne podejście i dużą bańkę bezczynności, gdy GPU oczekują na coś do zrobienia. Widzimy też lepszy sposób wykorzystania każdego urządzenia: wysyłanie danych w małych partiach. Mniejsza partia pozwala pierwszemu GPU szybciej przekazać to, nad czym pracował, i przejść do kolejnej partii. Dzięki temu następne urządzenie może rozpocząć pracę wcześniej, co w efekcie zmniejsza wielkość bańki.



Rysunek 3.6. *Problem bańki. Gdy dane przechodzą przez podzielony model, GPU przechowujące wagi modelu są niedostatecznie wykorzystywane i czekają na przetworzenie danych przez swoje odpowiedniki. Prostym sposobem na zmniejszenie tej bańki jest zastosowanie mniejszych partii danych*

Całkiem łatwo można obliczyć rozmiar bańki z użyciem następującego wzoru:

$$\text{Procent bezczynności} = 1 - m / (m + n - 1)$$

gdzie m to liczba mikrogrup, a n to głębokość potoku lub liczba GPU. W naszym naiwnym przykładzie z czterema GPU i jedną dużą partią widzimy, że urządzenia są bezczynne przez 75% czasu! Dopuszczenie do sytuacji, w której GPU będą bezczynnie oczekiwać przez trzy czwarte czasu, jest bardzo kosztowne. Zobaczmy, jak wygląda to przy zastosowaniu strategii mikrogrup. Przy mikrogrupie równej 4 czas bezczynności zmniejsza się prawie o połowę, do zaledwie 43%. Z tego wzoru możemy wywnioskować, że im więcej mamy GPU, tym dłuższe są czasy bezczynności, ale im więcej mikrogrup, tym wykorzystanie zasobów będzie lepsze.

Niestety, często nie możemy ani zmniejszyć liczby procesorów graficznych, ani zwiększyć rozmiaru mikrogrup do pożądanego poziomu. Istnieją bowiem pewne ograniczenia. Jeśli chodzi o GPU, to musimy użyć tylu, ilu potrzeba, aby zmieścić model w pamięci. Warto jednak starać się korzystać z kilku większych GPU, ponieważ prowadzi to do lepszego wykorzystania zasobów niż używanie wielu mniejszych kart graficznych. Zmniejszenie przestojów w zrównoleglaniu potokowym to kolejny powód, dla którego kompresja jest tak istotna. Jeśli chodzi o podział danych na mikrogrupy, to pierwsze ograniczenie jest oczywiste: ponieważ mikrogrupa jest częścią naszej grupy treningowej, jesteśmy ograniczeni jej rozmiarem. Drugie ograniczenie polega na tym, że każda mikrogrupa liniowo zwiększa zapotrzebowanie na pamięć dla buforowanych aktywacji. Jednym ze sposobów przeciwdziałania temu zwiększonemu zapotrzebowaniu na pamięć jest metoda zwana PipeDream[12]. Istnieją różne konfiguracje i podejścia, ale podstawowa idea pozostaje ta sama. W tej metodzie zaczynamy pracę nad propagacją wstecz, gdy tylko dla którejkolwiek z mikrogrup kończymy propagację w przód. Pozwala to na pełne ukończenie cyklu treningowego i zwolnienie pamięci podręcznej dla danej mikrogrupy.

ZRÓWNOLEGLENIE 3D

W przypadku dużych modeli językowych (LLM) będziemy chcieli wykorzystać wszystkie trzy praktyki zrównoleglania, ponieważ mogą one działać jednocześnie. Rozwiązanie to jest określane jako zrównoleglanie 3D, które łączy równoległe przetwarzanie danych, tensorów i potoków (DP + TP + PP). Ponieważ każda z technik, a tym samym każdy z wymiarów, wymaga do uruchomienia zrównoleglania 3D co najmniej dwóch GPU, to aby w ogóle można było myśleć o wykorzystaniu tej techniki, potrzebujemy minimum ośmiu GPU. Konfiguracja tych GPU będzie kluczowa w uzyskaniu maksymalnej wydajności całego procesu.

[12] A. Harlap i inni, *PipeDream: Fast and efficient pipeline parallel DNN training*, 8 czerwca 2018 r., *https://arxiv.org/abs/1806.03377*.

Ponieważ największe obciążenie komunikacyjne występuje w równoległości tensorowej (TP), należy zadbać, by używane GPU znajdowały się blisko siebie, najlepiej na tym samym węźle i maszynie. Zrównoleglanie potoków (PP) ma najmniejszą objętość komunikacji z tych trzech, więc podział modelu między węzłami jest tutaj najmniej kosztowny.

Uruchamiając te trzy techniki razem, można zaobserwować interesujące interakcje i synergie pomiędzy nimi. Ponieważ równoległość tensorowa dzieli model tak, aby dobrze mieścił się w pamięci urządzenia, widzimy, że zrównoleglanie potoków może działać dobrze nawet wtedy, gdy rozmiary grup są małe dzięki zastosowaniu równoległości tensorowej. Ta kombinacja poprawia również komunikację między węzłami przetwarzania danych na różnych etapach potoku, co pozwala na efektywne równoległe przetwarzanie danych.

Przepustowość komunikacji między węzłami jest proporcjonalna do liczby etapów potoku. W rezultacie nawet przy mniejszych rozmiarach grup równoległe przetwarzanie danych może się dobrze skalować. Ogólnie rzecz biorąc, widzimy, że jednoczesne uruchomienie pozwala uzyskać lepszą wydajność niż indywidualne stosowanie każdej z wymienionych technik.

Teraz gdy znasz już kilka sztuczek, równie ważne będzie wykorzystanie odpowiednich narzędzi, które pozwolą na wykonanie zadania.

3.4. Infrastruktura operacyjna modeli językowych

Wreszcie zaczynamy mówić o infrastrukturze niezbędnej do funkcjonowania dużych modeli językowych. Może to być zaskoczeniem, ponieważ niektórzy Czytelnicy spodziewali się tego podrozdziału już na początku książki. Dlaczego czekaliśmy z tym aż do końca rozdziału 3.?

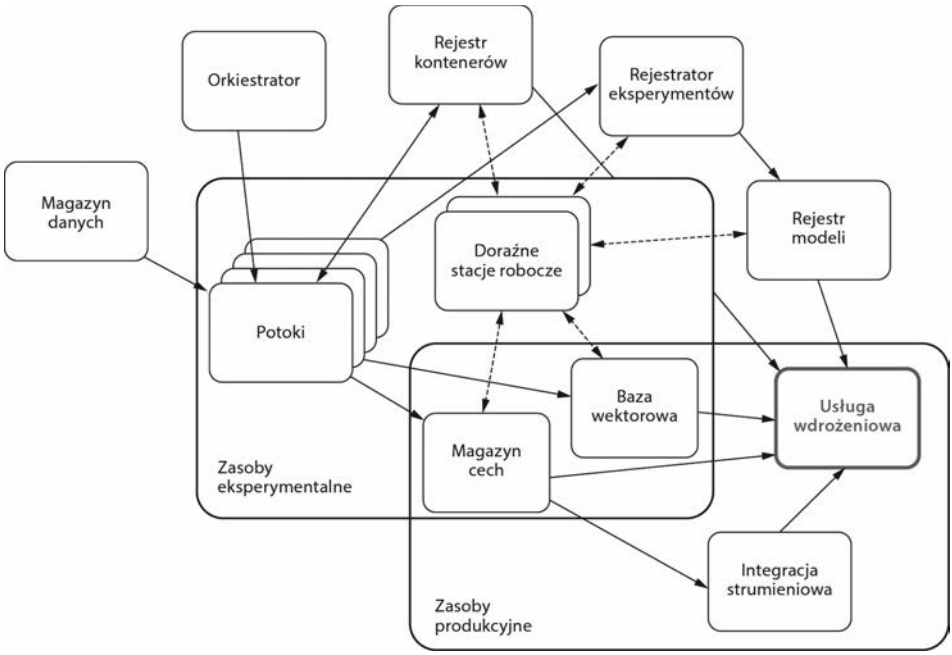
Podczas wielu rozmów kwalifikacyjnych z inżynierami uczenia maszynowego zadawaliśmy otwarte pytanie: „Co możesz powiedzieć o MLOps?”. To łatwe pytanie na rozgrzewkę. Większość młodszych kandydatów od razu zaczynała mówić o narzędziach i infrastrukturze. To zrozumiałe, jest tak wiele różnych dostępnych narzędzi. Już nie wspominając o tym, że gdy widzimy posty czy blogi opisujące MLOps, zawsze jest tam ładny diagram pokazujący infrastrukturę. Choć to wszystko jest ważne, warto zwrócić uwagę na to, o czym mówią bardziej doświadczeni kandydaci — o cyklu życia uczenia maszynowego.

Wielu osobom ta subtelna kwestia umyka, ale infrastruktura to „jak”, a cykl życia to „dlaczego”. Większość firm może sobie poradzić z minimalną infrastrukturą. Widzieliśmy wiele prowizorycznych systemów działających wyłącznie na laptopie jednego naukowca ds. danych, które działały zaskakująco dobrze — zwłaszcza w erze, gdy wszystko robiono z wykorzystaniem biblioteki scikit-learn!

Niestety, taka prowizoryczna platforma uczenia maszynowego nie sprawdza się w świecie modeli językowych. Skoro wciąż żyjemy w czasach, gdy standardowa pojemność dysku w MacBooku Pro to 256 GB, samo przechowywanie

modelu lokalnie może być problemem. Firmy inwestujące w solidniejszą infrastrukturę są znacznie lepiej przygotowane na świat modeli językowych.

Na rysunku 3.7 przedstawiona jest przykładowa infrastruktura MLOps zaprojektowana z myślą o modelach językowych. Choć większość diagramów infrastruktury upraszcza strukturę, aby wszystko wyglądało schludnie, brutalna prawda jest taka, że cały system jest nieco bardziej złożony. Oczywiście, wiele z tej złożoności mogłoby zniknąć, gdybyśmy mogli skłonić naukowców ds. danych do pracy w skryptach, a nie na doraźnych stacjach roboczych — zwykle z interfejsem Jupyter Notebook.



Rysunek 3.7. *Przegląd infrastruktury MLOps z uwzględnieniem dużych modeli językowych. Diagram ma na celu przedstawienie całościowego obrazu oraz złożoności licznych narzędzi, które wykorzystuje się do wdrażania modeli uczenia maszynowego w środowiskach produkcyjnych*

Dokładniejsza analiza rysunku 3.7 pozwala zauważyć na obrzeżach kilka narzędzi, które należą do obszaru DataOps lub nawet DevOps — magazyny danych, orkiestratory, potoki, integracje strumieniowe i rejestry kontenerów. To narzędzia, których prawdopodobnie już używasz w każdej aplikacji intensywnie wykorzystującej dane, niekoniecznie związanej z MLOps. W centrum schematu znajdują się bardziej tradycyjne narzędzia MLOps — śledzenie eksperymentów, rejestr modeli, magazyn cech i doraźne stacje robocze do analizy danych. Dla modeli językowych wprowadzamy właściwie tylko jedno nowe narzędzie: wektorową bazę danych. Na rysunku nie widać systemu monitorowania, ponieważ jest on powiązany z każdym elementem. To wszystko prowadzi do celu,

do którego dążymy w tej książce — usługi wdrożeniowej, w której z poczuciem pewności będziemy mogli wdrażać i uruchamiać modele językowe w środowisku produkcyjnym.

Infrastruktura według dyscyplin

Poniższa lista definiuje infrastrukturę z podziałem według poszczególnych dziedzin:

- *DevOps* — Odpowiada za pozyskiwanie zasobów środowiskowych: eksperymentalnych (programistycznych, testowych) i produkcyjnych. Obejmuje to sprzęt, klastry i infrastrukturę sieciową niezbędną do funkcjonowania całości. Zajmuje się również podstawowymi systemami infrastruktury, takimi jak repozytoria GitHub/GitLab, repozytoria artefaktów, rejestry kontenerów, bazy danych aplikacji lub transakcyjne bazy danych (np. Postgres czy MySQL), systemy buforowania oraz potoki CI/CD. Ta lista nie jest wyczerpująca.
- *DataOps* — Odpowiada za dane, zarówno w trakcie ich przekazywania, jak i przechowywania. Obejmuje scentralizowane lub rozproszone magazyny danych, takie jak hurtownie danych, magazyny typu data lake i sieci danych, a także potoki danych — zarówno w systemach wsadowych, jak i strumieniowych, wykorzystujących narzędzia takie jak Kafka i Flink. Zawiera również orkiestratory, takie jak Airflow, Prefect i Mage. DataOps opiera się na fundamencie DevOps. Przykładowo wiele potoków CI/CD było wykorzystywanych do pracy z potokami danych, zanim ostatecznie przeszły do systemów takich jak Apache Spark czy dbt.
- *MLOps* — Odpowiada za cykl życia uczenia maszynowego, od tworzenia modeli aż po ich wycofywanie. Obejmuje to stacje robocze do zadań związanych z nauką o danych, takie jak JupyterHub, narzędzia do śledzenia eksperymentów oraz rejestr modeli. W skład infrastruktury MLOps wchodzą także specjalistyczne bazy danych, takie jak magazyny cech i wektorowe bazy danych, jak również usługa wdrożeniowa, która łączy wszystkie elementy i dostarcza wyniki. MLOps opiera się na fundamencie zarówno DataOps, jak i DevOps.

W dalszej części rozdziału przyjrzyjmy się poszczególnym elementom układanki infrastrukturalnej i omówmy funkcje, które warto wziąć pod uwagę, myśląc o modelach LLM. Choć w przypadku każdego elementu będziemy omawiać specjalistyczne narzędzia, warto wspomnieć, że istnieją również platformy MLOps jako usługa, takie jak Dataiku, Amazon SageMaker, Azure Machine Learning czy Google VertexAI. Platformy te próbują zapewnić kompletne rozwiązanie; odrębną kwestią jest to, jak dobrze im się to udaje. Niemniej często stanowią one świetny skrót na drodze do celu, dlatego warto o nich wiedzieć. No dobrze, dość już tego owijania w bawełnę — zabierzmy się do rzeczy!

3.4.1. Infrastruktura danych

Choć nie jest to główny temat tej książki, warto zaznaczyć, że MLOps opiera się na infrastrukturze operacji na danych, która z kolei wspiera się na fundamencie DevOps. Kluczowe elementy ekosystemu DataOps to magazyn danych, orkiestrator

i potoki. Dodatkowo zwykle wymagane są rejestr kontenerów i usługa integracji strumieniowej.

Magazyny danych stanowią fundament DataOps i występują w wielu formach — od prostych baz danych po duże hurtownie danych, od jeszcze większych magazynów typu data lake po złożone siatki danych (ang. *data grid*). To tu przechowywane są dane, a wiele pracy poświęca się zarządzaniu magazynem danych oraz nadzorowaniu go i zabezpieczaniu. Orkiestrator jest kamieniem węgielnym DataOps, ponieważ automatyzuje zarówno proste, jak i złożone, wieloetapowe przepływy pracy i zadania oraz nimi zarządza, zapewniając ich wykonanie na wielu zasobach i usługach w systemie. Najczęściej wymieniane to Airflow, Prefect i Mage. Wreszcie potoki są filarami — podtrzymują całą resztę i to w nich uruchamiamy zadania. Choć początkowo stworzono je do przenoszenia, czyszczenia i definiowania danych, te same systemy wykorzystuje się do uruchamiania zadań uczenia maszynowego zgodnie z harmonogramem, wsadowego wykonywania wnioskowania i wielu innych prac niezbędnych do sprawnego funkcjonowania MLOps.

Rejestr kontenerów jest kluczowym elementem DevOps, a w konsekwencji także DataOps i MLOps. Uruchamianie wszystkich swoich potoków i usług w kontenerach jest niezbędne do zapewnienia spójności. Usługi strumieniowe to znacznie większe wyzwanie, niż można by sądzić na podstawie tego rozdziału — ci, którzy się tym zajmują, wiedzą, o co chodzi. Na szczęście w większości zadań związanych z przetwarzaniem tekstu przetwarzanie w czasie rzeczywistym nie jest głównym problemem. Nawet w zadaniach takich jak tworzenie napisów czy tłumaczenie w czasie rzeczywistym często możemy zastosować strategię przetwarzania quasirzeczywistego, która nie pogarsza doświadczenia użytkownika, choć to już może zależeć od konkretnego zadania.

3.4.2. Narzędzia do śledzenia eksperymentów

Narzędzia do śledzenia eksperymentów są kluczowe dla MLOps. Ich podstawowym zadaniem jest rejestrowanie testów i wyników. Jak stwierdził słynny Adam Savage z programu *Pogromcy mitów*: „Pamiętajcie, dzieciaki, jedyna różnica między zabawą a nauką polega na tym, że tę drugą się zapisuje". Bez tego Twoja organizacja prawdopodobnie pomija „naukę" w nauce o danych, co jest dość żenujące.

Nawet jeżeli naukowcy ds. danych skrupulatnie zapisują wyniki w notatnikach, to i tak jest to bezużyteczne, o ile inni nie będą mogli ich łatwo przeglądać i przeszukiwać zapisanych w nich informacji. To właśnie jest głównym celem narzędzi do śledzenia eksperymentów — mają one zapewniać, że wiedza jest łatwo dostępna i można się nią dzielić. Prędzej czy później model trafi do produkcji i pojawią się problemy. Owszem, zawsze można wytrenować nowy model, ale jeśli zespół nie będzie w stanie wrócić i zbadać, co poszło nie tak za pierwszym razem, prawdopodobnie będzie powtarzać te same błędy.

Istnieje wiele narzędzi do śledzenia eksperymentów. Zdecydowanie najpopularniejszym jest stworzony przez zespół Databricks otwartoźródłowy MLFlow, który oferuje również łatwe rozwiązanie hostingowe. Wśród płatnych alternatywnych opcji warte uwagi są CometML i Weights & Biases.

Współczesne narzędzia do śledzenia eksperymentów mają wiele dodatkowych funkcji. Większość rozwiązań open source i płatnych z pewnością zaspokoi Twoje potrzeby w zakresie LLMOps. Jednak aby w pełni wykorzystać te narzędzia, być może trzeba będzie wprowadzić kilka drobnych modyfikacji. Na przykład zazwyczaj domyślnie zakłada się, że modele są trenowane od podstaw, ale pracując z LLM-ami, często będziesz dostrajać już istniejące modele. W takim przypadku ważne jest odnotowanie punktu kontrolnego modelu, od którego zacząłeś. Jeśli to możliwe, warto nawet zapisać odnośnik do pierwotnego eksperymentu treningowego. Pozwoli to przyszłym naukowcom głębiej analizować wyniki testów, odnajdywać początkowe dane treningowe i odkrywać sposoby na wyeliminowanie stronniczości.

Inną przydatną funkcją są narzędzia do oceny metryk. Zagadnieniem metryk zajmiemy się bardziej szczegółowo w rozdziale 4., ale warto wspomnieć, że metryki ewaluacyjne dla modeli językowych są skomplikowane. Często interesuje nas wiele metryk, a żadna z nich nie jest prosta, choćby oceny złożoności czy wskaźniki podobieństwa. Wprawdzie dostawcy narzędzi do śledzenia eksperymentów starają się zachować neutralność i nie narzucać własnych opinii co do metryk ewaluacyjnych, ale powinni przynajmniej ułatwiać porównywanie modeli i ich metryk, aby pomóc nam zdecydować, który z nich jest lepszy. Odkąd modele LLM stały się tak popularne, niektórzy dodali obsługę powszechnych metryk, takich jak ROUGE do podsumowywania tekstu.

Zauważysz także, że wielu dostawców narzędzi do śledzenia eksperymentów zaczęło dodawać funkcje specyficzne dla dużych modeli językowych. Warto zwrócić uwagę na takie elementy jak bezpośrednia obsługa Hugging Face, integracja z LangChain, narzędzia do inżynierii promptów, frameworki do dostrajania oraz biblioteki modeli bazowych. Ta dziedzina rozwija się bardzo szybko i obecnie żadne narzędzie nie udostępnia wszystkich tych funkcji, ale prawdopodobnie z czasem nastąpi konwergencja tych zestawów funkcjonalności.

3.4.3. Rejestr modeli

Rejestr modeli jest prawdopodobnie najprostszym narzędziem infrastruktury MLOps. Jego głównym celem jest coś łatwego do zrealizowania — potrzebujemy po prostu miejsca do przechowywania modeli. Wiele odnoszących sukcesy zespołów radzi sobie w ten sposób, że umieszcza swoje modele w magazynie obiektów lub współdzielonym systemie plików, i uznaje to za wystarczające rozwiązanie. Niemniej jednak wybierając rejestr modeli, warto zwrócić uwagę na kilka dodatkowych funkcji.

Po pierwsze, należy sprawdzić, czy rejestr śledzi metadane modelu. Większość istotnych informacji znajduje się zazwyczaj w narzędziu do śledzenia eksperymentów, więc zwykle wystarczy upewnić się, że można połączyć te dwa elementy. W rzeczywistości większość rejestrów modeli jest wbudowana w systemy śledzenia eksperymentów właśnie z tego powodu. Jednak problem z tymi systemami pojawia się, gdy firma decyduje się na wykorzystanie modelu open source lub nawet zakup gotowego modelu. Czy w takich przypadkach można łatwo wgrać model i oznaczyć go odpowiednimi informacjami? Zazwyczaj odpowiedź na to pytanie jest przecząca.

Następnie warto się upewnić, że można wersjonować modele. W pewnym momencie model może przestać być użyteczny i będzie wymagał zastąpienia. Możliwość wersjonowania modeli uprości ten proces. Ułatwia to również przeprowadzanie eksperymentów produkcyjnych, takich jak testy A/B czy testy równoległe (ang. *shadow tests*).

Wreszcie, jeśli przenosimy modele do środowiska produkcyjnego lub wycofujemy je, musimy zadbać o kontrolę dostępu. Modele dla wielu firm stanowią cenną własność intelektualną, dlatego ważne jest, aby tylko odpowiedni użytkownicy mieli do nich dostęp. Równie istotne jest zapewnienie, że tylko zespół rozumiejący modele — ich działanie i powody, dla których zostały wytrenowane — będzie odpowiedzialny za ich przenoszenie pomiędzy środowiskami lub wycofywanie. Ostatnią rzeczą, jakiej chcemy, jest przypadkowe usunięcie modelu produkcyjnego lub coś jeszcze gorszego.

W przypadku dużych modeli językowych należy pamiętać o kilku ważnych kwestiach: przede wszystkim wybierając rejestr modeli, trzeba zwrócić uwagę na ewentualne ograniczenia rozmiaru. Niektóre rejestry modeli ograniczają rozmiar modelu do 10 GB lub mniej. To zdecydowanie za mało. Możemy spekulować na temat wielu powodów takiego stanu rzeczy, ale żaden z takich rejestrów nie jest wart uwagi. Mówiąc o ograniczeniach rozmiaru, jeśli zamierzasz uruchomić rejestr modeli w lokalnym systemie przechowywania, takim jak Ceph, upewnij się, że ma on dużo miejsca. Możesz kupić wiele terabajtów pamięci masowej za kilkaset złotych dla swoich serwerów lokalnych, ale nawet kilka terabajtów szybko się zapełni, gdy Twój model LLM przekracza 300 GB. Pamiętaj: podczas trenowania i dostrajania modeli prawdopodobnie będziesz, w celu zapewnienia niezawodności, przechowywać wiele punktów kontrolnych i wersji, a także ich duplikaty. Jednak pamięć masowa to nadal jeden z najtańszych aspektów uruchamiania modeli LLM, więc nie ma powodu, aby na niej oszczędzać i powodować problemy w przyszłości.

To prowadzi do ważnej kwestii: wciąż można wprowadzić wiele optymalizacji, umożliwiających bardziej efektywne oszczędzanie miejsca przy przechowywaniu modeli LLM i ich pochodnych, zwłaszcza że większość tych modeli będzie ogólnie bardzo podobna. Prawdopodobnie w przyszłości pojawią się rozwiązania do przechowywania danych, które rozwiążą ten problem.

3.4.4. Magazyny cech

Magazyny cech rozwiązują wiele istotnych problemów i odpowiadają na pytania takie jak: Kto jest właścicielem tej cechy? Jak została zdefiniowana? Kto ma do niej dostęp? Które modele jej używają? Jak udostępnić tę cechę w środowisku produkcyjnym? W zasadzie rozwiązują one problem „pojedynczego źródła prawdy". Stworzenie scentralizowanego magazynu pozwala zespołom na wybór najwyższej jakości, najlepiej utrzymanych i dokładnie zarządzanych danych. Magazyny cech rozwiązują problemy związane ze współpracą, dokumentacją i wersjonowaniem danych.

Jeśli kiedykolwiek pomyślałeś: „Magazyn cech to po prostu baza danych, prawda?", prawdopodobnie myślisz o niewłaściwym rodzaju magazynu — my odnosimy się do miejsca, w którym robi się zakupy, a nie miejsca przechowywania. Nie martw się, to normalne, że tak myślisz — często słyszeliśmy podobne opinie i sami mieliśmy podobne przemyślenia. Prawda jest taka, że współczesne magazyny cech są bardziej wirtualne niż fizyczne bazy danych, co oznacza, że buduje się je na fundamencie dowolnego magazynu danych, którego już używasz. Na przykład magazyn cech Google Vertex AI to po prostu BigQuery, a my widzieliśmy wiele nieporozumień wśród zespołów ds. danych, które zastanawiały się: „Dlaczego po prostu nie zapytać BigQuery?". Ładowanie danych do magazynu cech może wydawać się niepotrzebnym dodatkowym krokiem, ale wyobraź sobie zakupy w sklepie IKEA. Nikt nie idzie bezpośrednio do magazynu, gdzie leżą wszystkie meble zapakowane w pudełkach. To byłoby frustrujące doświadczenie. Magazyn cech to salon wystawowy, który pozwala innym osobom w firmie łatwo przeglądać dane, analizować je oraz z nich korzystać.

Często widzimy, jak ludzie sięgają po magazyn cech, aby rozwiązać problem techniczny, taki jak dostęp do cech dostępnych online z niskimi opóźnieniami. Ogromną zaletą magazynów cech jest to, że rozwiązują one problem rozbieżności pomiędzy trenowaniem a wdrażaniem. Niektóre cechy można po prostu łatwiej obliczyć po jakiejś operacji, używając do tego języka SQL; przykładem może tu być obliczenie średniej liczby zapytań z ostatnich 30 sekund. Może to prowadzić do tworzenia bardzo prostych potoków danych treningowych, lecz jednocześnie powodować ogromne problemy przy wdrażaniu rozwiązania w środowisku produkcyjnym, ponieważ uzyskanie tego typu cechy w czasie rzeczywistym może nastręczać poważnych trudności. Abstrakcje magazynu cech pomagają zminimalizować te trudności. Kolejnym powiązanym zagadnieniem jest pobieranie cech z określonego momentu; niezbędne do tego możliwości funkcjonalne są standardem w przypadku magazynów cech. Pobieranie danych z określonego punktu czasu sprawia, że dla konkretnego momentu zapytanie zawsze zwróci taki sam wynik. Ma to istotne znaczenie, ponieważ cechy takie jak średnie z „ostatnich 30 sekund" stale się zmieniają, więc możliwość pobierania danych z określonego momentu pozwala na wersjonowanie danych (bez

dodatkowego obciążenia rozbudowanym systemem wersjonowania), a także zapewnia, że nasze modele będą dawać dokładne i przewidywalne odpowiedzi.

Jeśli chodzi o dostępne opcje, to najpopularniejszym otwartoźródłowym magazynem cech jest obecnie Feast. Inne rozwiązania open-source to Featureform i Hopsworks. Wszystkie trzy oferują płatne opcje hostingu. W kontekście dużych modeli językowych spotkaliśmy się z opiniami, że magazyny cech nie są tak krytyczne jak inne części infrastruktury MLOps. W końcu model jest tak duży, że powinien zawierać wszystkie potrzebne cechy, więc nie trzeba pytać o dodatkowy kontekst. Wystarczy podać modelowi zapytanie użytkownika i pozwolić mu działać. Jednak to podejście nadal jest nieco naiwne i nie doszliśmy jeszcze do punktu, w którym modele językowe są całkowicie samowystarczalne. Aby uniknąć halucynacji i poprawić faktyczną poprawność, często najlepiej jest dostarczyć modelowi pewien kontekst. Robimy to przez podanie mu osadzeń naszych dokumentów, których znajomości od niego oczekujemy, a magazyn cech jest świetnym miejscem do przechowywania tych osadzeń.

3.4.5. Wektorowe bazy danych

Jeśli znasz ogólną infrastrukturę MLOps, większość tego punktu będzie dla Ciebie jedynie przypomnieniem tego, co już wiesz. Musieliśmy jedynie wprowadzić niewielkie modyfikacje, uwzględniające istotne kwestie skalowania, co było niezbędne, by system działał z modelami LLM. Wektorowe bazy danych są jednak nowym rozwiązaniem, stworzonym specjalnie do pracy z modelami LLM oraz ogólnie pojętymi modelami językowymi. Można je również wykorzystać do pracy z innymi zbiorami danych, takimi jak obrazy czy dane tabelaryczne, które łatwo przekształcić w wektory. Wektorowe bazy danych to wyspecjalizowane bazy przechowujące wektory wraz z metadanymi. Dzięki temu idealnie się nadają do przechowywania osadzeń. Choć to stwierdzenie jest prawdziwe, może być nieco mylące, ponieważ siła wektorowych baz danych tkwi nie w możliwościach przechowywania danych wektorowych, ale w możliwościach ich przeszukiwania.

Tradycyjne bazy danych, wykorzystujące indeksy typu *b-tree* do wyszukiwania identyfikatorów lub wyszukiwania tekstowego z użyciem indeksów odwróconych, mają tę samą wadę: musimy wiedzieć, czego szukamy. Jeśli nie znamy identyfikatora lub słów kluczowych, znalezienie właściwego wiersza czy dokumentu jest niemożliwe. Natomiast wektorowe bazy danych wykorzystują przestrzeń wektorową, co oznacza, że nie musimy dokładnie wiedzieć, czego szukamy; wystarczy znać coś podobnego, co pozwoli znaleźć najbliższych sąsiadów za pomocą wyszukiwania podobieństwa opartego na odległości euklidesowej, podobieństwie cosinusowym, iloczynowym lub innym. Na przykład użycie wektorowej bazy danych znacznie ułatwia rozwiązanie problemu wyszukiwania obrazów.

W tym momencie niektórzy Czytelnicy mogą być zdezorientowani. Najpierw powiedzieliśmy, że osadzenia należy umieścić w magazynie cech, a teraz mówimy o umieszczeniu ich w bazie danych wektorowych. Które rozwiązanie jest właściwe? Otóż najlepsze jest to, że można je zastosować jednocześnie. Jeśli wcześniej nie miało to sensu, mamy nadzieję, że teraz zalety takiego rozwiązania stają się jasne. Magazyn cech nie jest bazą danych; to tylko abstrakcja. Można użyć magazynu cech opartego na danych wektorowych, co rozwiąże wiele problemów. Wektorowe bazy danych mogą być trudne do utrzymania, gdy mamy wiele źródeł danych, eksperymentujemy z różnymi modelami osadzania lub często aktualizujemy dane. Zarządzanie tą złożonością może być prawdziwym wyzwaniem, ale magazyn cech skutecznie rozwiązuje ten problem. Jednoczesne stosowanie obu tych narzędzi sprawi, że będziemy dysponować dokładniejszym i aktualnym indeksem wyszukiwania.

W czasie kiedy pisaliśmy tę książkę, wektorowe bazy danych istniały zaledwie od kilku lat i dopiero zyskują popularność, gdyż rozwijały się równolegle z modelami LLM. Łatwo zrozumieć przyczyny tego stanu rzeczy, ponieważ zapewniają szybki i efektywny sposób pobierania danych wektorowych, co ułatwia dostarczanie modelom LLM niezbędnego kontekstu pozwalającego na poprawianie dokładności wyników.

To stosunkowo nowa dziedzina, lecz i tak istnieje wiele konkurujących ze sobą rozwiązań. Jest zbyt wcześnie, aby wskazać zwycięzców i przegranych. Mając na względzie dbałość o zachowanie aktualności informacji zamieszczonych w tej książce, pozwolimy sobie zasugerować na początek dwie opcje: Pinecone i Milvus. Pinecone jest jedną z pierwszych wektorowych baz danych oferowanych jako produkt i ma prężną społeczność oraz obszerną dokumentację. Udostępnia bogate funkcjonalności, a jego skalowalność potwierdzono w praktyce. Pinecone to w pełni zarządzana oferta infrastrukturalna, która udostępnia darmowy plan dla początkujących. Jeśli jednak jesteś zwolennikiem oprogramowania otwartoźródłowego, warto, żebyś zainteresował się Milvusem. Milvus także oferuje bogate możliwości funkcjonalne i ma świetną społeczność. Zilliz, firma, która stworzyła Milvusa, oferuje w pełni zarządzane rozwiązanie, ale można go również wdrożyć we własnych klastrach. Jeśli masz już pewne doświadczenie w zarządzaniu infrastrukturą, to przygotowanie odpowiedniego środowiska nie powinno przysporzyć Ci problemów.

Obecnie istnieje wiele alternatywnych opcji i przed dokonaniem wyboru warto poświęcić trochę czasu na ich zbadanie. Dwoma czynnikami, na które prawdopodobnie zwrócisz największą uwagę, są cena i skalowalność, ponieważ często są one ze sobą powiązane. Następnie warto dokładniej przyjrzeć się funkcjom wyszukiwania, a w szczególności możliwościom stosowania różnych miar podobieństwa, takich jak podobieństwo cosinusowe, iloczyn skalarny i odległość euklidesowa; kolejnym aspektem są dostępne funkcje indeksowania, takie jak Hierarchical Navigable Small World (HNSW) i locality-sensitive hashing

(LSH). Możliwość dostosowania parametrów wyszukiwania i ustawień indeksu jest ważna dla każdej bazy danych, ponieważ pozwala na optymalizację obciążenia dla konkretnego zestawu danych i przepływu pracy, co umożliwia optymalizację opóźnień zapytań i dokładności wyników wyszukiwania.

Warto również zauważyć, że wraz z rosnącą popularnością wektorowych baz danych wielu dotychczasowych dostawców baz, takich jak Redis i Elastic, oferuje możliwości wyszukiwania wektorowego. Na razie większość z nich udostępnia najprostsze zestawy funkcji, ale jeśli już korzystasz z tych narzędzi, to trudno je zignorować, ponieważ mogą zapewnić szybkie korzyści i pomóc w szybkim rozpoczęciu pracy.

Wektorowe bazy danych są potężnymi narzędziami, które mogą pomóc w trenowaniu lub dostrajaniu modeli LLM, a także poprawić dokładność i wyniki zapytań do tych modeli.

3.4.6. System monitorowania

System monitorowania jest kluczowy dla sukcesu każdego systemu uczenia maszynowego, w tym modeli językowych. W przeciwieństwie do innych aplikacji modele ML mogą zawodzić w sposób niezauważalny — nadal działają, ale zaczynają zwracać gorsze wyniki. Często jest to spowodowane zjawiskiem dryfu danych, czego typowym przykładem jest system rekomendacji, który z czasem daje coraz gorsze rezultaty, ponieważ sprzedawcy zaczynają nim manipulować przez dodawanie fałszywych recenzji w celu uzyskania lepszych rekomendacji. System monitorowania pozwala wychwycić słabo działające modele i wprowadzić do nich poprawki lub po prostu je ponownie wytrenować.

Mimo swojego znaczenia systemy monitorowania w wielu przypadkach są ostatnim elementem układanki. Często jest to celowa decyzja, wynikająca z faktu, że inwestowanie zasobów w opracowanie sposobów monitorowania modeli jest pozbawione sensu, jeśli nie ma jeszcze żadnych modeli do monitorowania. Nie należy jednak popełniać błędu i zbyt długo odkładać uruchomienie takiego systemu. Wiele firm ucierpiało z powodu modelu, który wymknął się spod kontroli, o czym nikt nie wiedział, co naraziło firmę na poważne koszty. Ważne jest też, aby zdać sobie sprawę, że z monitorowaniem nie trzeba czekać aż do wdrożenia modelu. Istnieje wiele sposobów na wprowadzenie systemu monitorowania do procesów trenowania i przetwarzania danych, co poprawia zarządzanie danymi i zgodność z przepisami. Niezależnie od tego dojrzałość organizacji zajmującej się nauką o danych zwykle można ocenić po jej systemie monitorowania.

Istnieje wiele świetnych narzędzi do monitorowania. Do szczególnie godnych polecenia rozwiązań otwartoźródłowych tego typu należy zaliczyć WhyLogs i Evidently AI. Jesteśmy też gorącymi zwolennikami Great Expectations, ale uznaliśmy, że poza zadaniami wsadowymi działa dość wolno. Istnieje również wiele rozwiązań komercyjnych. Monitorując systemy ML, zazwyczaj warto śledzić wszystko, co normalnie rejestruje się w innych aplikacjach. Obejmuje to

metryki zasobów, takie jak wykorzystanie pamięci i procesora, metryki wydajności, takie jak opóźnienia i liczba zapytań na sekundę, oraz metryki operacyjne, takie jak kody statusu i wskaźniki błędów. Ponadto będzie trzeba zapewnić możliwość monitorowania dryfu danych wchodzących do modelu i z niego wychodzących. Warto zwrócić uwagę na takie rzeczy jak brakujące wartości, unikalność i zmiany wartości odchylenia standardowego. W wielu przypadkach przydatna będzie możliwość segmentacji danych podczas monitorowania — na przykład na potrzeby testów A/B lub monitorowania według regionów. Niektóre przydatne metryki do monitorowania w systemach ML to: dokładność modelu, precyzja, czułość i wynik F1. Są one trudne do zmierzenia, ponieważ nie znamy poprawnej odpowiedzi już w czasie wnioskowania, dlatego często pomocne jest stworzenie systemu audytu. Oczywiście audyt będzie łatwiejszy, jeśli model językowy jest zaprojektowany jako bot do pytań i odpowiedzi, a nie jako narzędzie do wspomagania kreatywności pisarzy.

To wskazuje na całkiem nowy zestaw wyzwań stawianych przed systemami monitorowania, nawet większych niż przed innymi systemami ML. W przypadku modeli językowych mamy do czynienia z danymi tekstowymi, które, jak już wspominaliśmy wcześniej w tym rozdziale, jest bardzo trudno ocenić jakościowo. Na przykład zastanów się nad cechami analizowanymi w celu monitorowania dryfu danych, co jest ważne, gdyż język znany jest z ciągłych zmian! Jedną z cech, które sugerujemy, są unikalne tokeny. Będą one ostrzegać, gdy pojawią się nowe slangowe słowa lub terminy. Jednak same tokeny w niczym nam nie pomogą w przypadku, gdy słowa zmieniają znaczenie, na przykład gdy „zarąbisty” zaczął oznaczać „świetny”. Zalecalibyśmy również, by w razie pojawienia się problemów zdecydować się na monitorowanie osadzeń, jednak prawdopodobnie okaże się, że dodaje to dużo szumu i fałszywych alarmów, a przynajmniej jest trudne do rozszyfrowania i zbadania. Najlepiej działające systemy często uwzględniają wiele ręcznie tworzonych reguł i cech, które są monitorowane, jednak takie systemy mogą być podatne na błędy i czasochłonne w tworzeniu.

Monitorowanie systemów opartych na tekście w żadnym razie nie pozwala uznać, że problem jest rozwiązany, głównie ze względu na trudności, których przysparza rozumienie danych tekstowych. Rodzi to pytanie o najlepsze metody wykorzystania modeli językowych do monitorowania samych siebie, ponieważ to właśnie modele LLM są obecnie naszym najlepszym rozwiązaniem do kodyfikacji języka. Niestety, nie znamy nikogo, kto by to badał, ale wyobrażamy sobie, że to tylko kwestia czasu.

3.4.7. Stacje robocze z obsługą GPU

Stacje robocze wyposażone w procesory graficzne oraz zdalne stacje robocze wiele zespołów postrzega jako miły dodatek albo luksus, ale w pracy z modelami językowymi (LLM) to podejście musi się zmienić. Podczas rozwiązywania problemów lub ogólnego rozwijania modelu naukowiec ds. danych nie jest już

w stanie uruchomić modelu w notatniku Jupyter na swoim laptopie. Najprostszym rozwiązaniem jest zapewnienie zdalnych stacji roboczych z zasobami GPU. Istnieje wiele rozwiązań chmurowych w tym zakresie, ale jeśli firma działa głównie lokalnie, zapewnienie takich zasobów może być trudniejsze, choć nadal konieczne.

Modele LLM intensywnie wykorzystują pamięć GPU. W związku z tym każdy inżynier pracujący w tej dziedzinie powinien znać pewne kluczowe liczby. Pierwszą z nich jest liczba potrzebnych GPU. NVIDIA Tesla T4 i V100 to dwa najpopularniejsze układy GPU, które można znaleźć w centrach danych, ale mają one tylko 16 GB pamięci. Są to jednak wydajne i opłacalne rozwiązania, więc jeśli uda nam się skompresować model tak, aby działał na tych układach — świetnie. Poza nimi dostępna jest gama GPU, takich jak NVIDIA A10G, seria NVIDIA Quadro i seria NVIDIA RTX, oferujących pamięć GPU w rozmiarze 24, 32 i 48 GB. Wszystkie te układy to dobre ulepszenie; trzeba tylko sprawdzić, które z nich są oferowane przez dostawcę usług chmurowych i dostępne dla nas. To prowadzi nas do modelu NVIDIA A100, który prawdopodobnie będzie preferowanym GPU do pracy z modelami LLM. Na szczęście układy te są stosunkowo powszechne i oferowane w dwóch wariantach: 40 GB lub 80 GB. Dużym problemem jest to, że obecnie zapotrzebowanie na nie jest ogromne. Warto również wiedzieć o NVIDIA H100, który udostępnia 80 GB pamięci, podobnie jak A100. Producent zapowiada, że model H100 NVL, zaprojektowany z myślą o modelach LLM, będzie obsługiwać do 188 GB pamięci. Innym nowym GPU, o którym warto wiedzieć, jest NVIDIA L4 Tensor Core GPU, który ma 24 GB pamięci i ma zastąpić T4 i V100 jako nowy „koń roboczy”, przynajmniej jeśli chodzi o zadania związane z AI.

Modele LLM mają różne rozmiary i warto mieć ogólne pojęcie o tym, co oznaczają te liczby. Na przykład model Llama ma warianty o 7, 13, 33 i 65 miliardach parametrów. Jeśli nie jesteś pewien, jakiego GPU potrzebujesz do uruchomienia danego modelu, oto prosty sposób: pomnóż liczbę miliardów parametrów przez dwa, a otrzymasz wymaganą ilość pamięci GPU. Wynika to z faktu, że większość modeli podczas wnioskowania domyślnie działa z zastosowaniem połowy precyzji, FP16 lub BF16, co oznacza, że potrzebujemy co najmniej 2 bajtów na każdy parametr. Na przykład 7 miliardów × 2 bajty = 14 GB. Potrzeba też trochę dodatkowej pamięci na model osadzania, co zajmie około gigabajta, oraz nieco więcej na same tokeny przetwarzane przez model. Jeden token to około 1 MB, więc 512 tokenów wymaga 512 MB. Nie jest to duży problem, dopóki nie weźmiemy pod uwagę większych rozmiarów partii danych (ang. *batch*) stosowanych w celu poprawy wydajności. Dla 16 takich partii danych potrzeba zatem dodatkowych 8 GB przestrzeni.

Oczywiście, do tej pory mówiliśmy tylko o wnioskowaniu; do trenowania potrzeba znacznie więcej miejsca. Podczas trenowania zawsze chcemy pracować z pełną precyzją, a dodatkowo potrzebujemy miejsca na tensory optymalizatora

i gradienty. Ogólnie rzecz biorąc, aby to uwzględnić, potrzeba około 16 bajtów na parametr. Zatem do trenowania modelu z 7B parametrów potrzeba 112 GB pamięci.

3.4.8. Usługa wdrożeniowa

Załóżmy, że wszystko, nad czym pracowaliśmy, zostaje zebrane i wreszcie wykorzystane w praktyce. W rzeczywistości, nawet gdybyśmy pozbyli się wszystkich innych usług i zostali tylko z usługą wdrożeniową, nadal mielibyśmy działający system MLOps. Usługa wdrożeniowa zapewnia łatwą integrację ze wszystkimi wcześniej omówionymi systemami oraz konfigurację i definiowanie zasobów niezbędnych do uruchomienia modelu w środowisku produkcyjnym. Często dostarcza ona gotowy kod służący do udostępniania modelu za pośrednictwem interfejsów API REST i gRPC albo bezpośrednio w potoku wsadowym lub strumieniowym.

Narzędzia pomocne do tworzenia takich usług to między innymi NVIDIA Triton Inference Service, MLServer, Seldon i BentoML. Usługi te zapewniają standardowy interfejs API, zazwyczaj zgodny z protokołem KServe V2 Inference. Protokół ten oferuje ujednolicony i rozszerzalny sposób wdrażania i udostępniania modeli uczenia maszynowego na różnych platformach i w różnych frameworkach oraz zarządzania nimi. Definiuje on wspólny interfejs do interakcji z modelami, w tym API gRPC i HTTP/REST. Standaryzuje koncepcje takie jak kodowanie wejściowych/wyjściowych danych tensorowych, metody przewidywania i wyjaśniania, kontrole stanu modelu oraz pobieranie metadanych. Umożliwia również bezproblemową integrację z różnymi językami i frameworkami, w tym TensorFlow, PyTorch, ONNX, Scikit-learn i XGBoost.

Oczywiście, czasami elastyczność i możliwość dostosowania są na tyle cenne, że warto odejść od zautomatyzowanej ścieżki oferowanej przez te frameworki. W takich przypadkach najlepiej sięgnąć po narzędzie takie jak FastAPI. Usługa wdrażania powinna jednak nadal zapewniać jak najwięcej automatyzacji i gotowego kodu, aby proces był jak najbardziej płynny. Warto wspomnieć, że większość wymienionych wcześniej frameworków oferuje również metody niestandardowe, ale ich skuteczność może być różna.

Wdrożenie modelu to więcej niż tylko zbudowanie interfejsu. Usługa wdrażania stanowi również pomost między infrastrukturą MLOps a ogólną infrastrukturą DevOps. Zapewnia połączenie z narzędziami CI/CD oraz potokami budowania i dostarczania, które firma ma skonfigurowane, dzięki czemu można łatwo monitorować i przeprowadzać odpowiednie testy oraz strategie wdrażania, takie jak kontrole stanu i wycofywanie zmian. Stosowane rozwiązania są bardzo specyficzne dla danej platformy, a tym samym dla firmy. Muszą także zapewniać niezbędne ustawienia konfiguracyjne do komunikacji z Kubernetesem lub innym orkiestratorem kontenerów, aby pozyskać potrzebne zasoby, takie jak CPU, pamięć, akceleratory, mechanizmy do automatycznego skalowania (autoskalery),

serwery proxy itp. Stosuje też niezbędne zmienne środowiskowe i narzędzia do zarządzania sekretami, aby zapewnić prawidłowe działanie wszystkich elementów.

Podsumowując, usługa ta gwarantuje, że można łatwo wdrożyć model w środowisku produkcyjnym. W przypadku modeli LLM głównym problemem często jest konieczność upewnienia się, że platforma i klastry są skonfigurowane z wystarczającą ilością zasobów do obsługi tego, co ostatecznie będzie na nich uruchamiane.

W tym rozdziale omówiliśmy wiele zagadnień, począwszy od tego, co sprawia, że modele LLM są znacznie trudniejsze niż tradycyjne uczenie maszynowe, które samo w sobie jest już wystarczająco skomplikowane. Najpierw dowiedziałeś się, że ich rozmiar jest czynnikiem, którego nie można nie doceniać, a następnie poznałeś wiele osobliwości modeli LLM, od limitów tokenów po halucynowanie — już nie wspominając o tym, że są one bardzo kosztowne. Na szczęście, pomimo wszelkich trudności, tworzenie i stosowanie modeli LLM można opanować. W rozdziale przedstawione zostały także techniki kompresji i przetwarzania rozproszonego, których poznanie i opanowanie ma kluczowe znaczenie. Następnie przyjrzeliśmy się infrastrukturze niezbędnej do funkcjonowania modeli LLM. Chociaż prawdopodobnie już wcześniej znałeś jej kluczowe elementy, to jednak zwróciliśmy uwagę, że modele LLM wywierają znacznie silniejszą presję na każde z tych narzędzi, i często musimy być gotowi na większą skalę niż ta, którą moglibyśmy się zadowolić przy wdrażaniu innych modeli ML.

Podsumowanie

- Duże modele językowe są trudne w obsłudze głównie ze względu na ich rozmiar. Przekłada się to na dłuższy czas pobierania, ładowania do pamięci i wdrażania oraz zmusza nas do korzystania z kosztownych zasobów.
- LLM-y są również skomplikowane, ponieważ mają do czynienia z językiem naturalnym i całą jego złożonością, w tym halucynacjami, stronniczością, kwestiami etycznymi i bezpieczeństwem.
- Niezależnie od tego, czy budujemy własne rozwiązanie, czy korzystamy z gotowego, LLM-y są kosztowne, a zarządzanie wydatkami i ryzykiem, jakie wiążą się z ich wykorzystaniem, będzie miało fundamentalne znaczenie dla powodzenia każdego projektu opartego na tych modelach.
- Kompresja modelu do jak najmniejszych rozmiarów ułatwia korzystanie z niego; szczególnie przydatne do tego celu są: kwantyzacja, przycinanie i destylacja wiedzy.
- Kwantyzacja jest popularna, ponieważ jest łatwa i można ją przeprowadzić po wytrenowaniu modelu bez konieczności jego dostrajania.

- Aproksymacja niskiego rzędu to skuteczny sposób na zmniejszenie modelu i, dzięki metodzie LoRA, powszechnie stosowany.
- Istnieją trzy główne kierunki równoległego przetwarzania w LLM: równoległe przetwarzanie danych, tensorów i potoku. Równoległe przetwarzanie danych pomaga zwiększyć przepustowość, równoległe przetwarzanie tensorów przyspiesza obliczenia, a równoległe przetwarzanie potoku umożliwia uruchomienie całego procesu.
- W wyniku połączenia metod równoległego przetwarzania otrzymujemy przetwarzanie trójwymiarowe (dane + tensory + potok), w którym wszystkie trzy techniki uzupełniają się wzajemnie, kompensując wzajemnie swoje słabe strony i pomagając zapewnić lepsze wykorzystanie zasobów.
- Infrastruktura dla LLMOps jest podobna do tej dla MLOps, ale nie dajmy się zwieść — istnieje wiele niuansów, kiedy „wystarczająco dobre" rozwiązania przestają wystarczać.
- Wiele narzędzi oferuje obecnie nowe możliwości dostosowane szczególnie do obsługi dużych modeli językowych.
- Szczególnie interesującym nowym elementem infrastruktury niezbędnej do tworzenia i obsługi modeli językowych są wektorowe bazy danych, które umożliwiają szybkie wyszukiwanie i pobieranie wektorów cech.

Od teorii do produkcji — praktyczny przewodnik po wdrażaniu LLM

Duże modele językowe (LLM) rewolucjonizują branżę IT, oferując bezprecedensowe możliwości w zakresie przetwarzania języka naturalnego. ChatGPT i podobne rozwiązania pokazały ogromny potencjał tej technologii, ale wdrożenie LLM w środowiskach produkcyjnych to znacznie więcej niż tylko korzystanie z gotowych API. Książka wypełnia lukę między teorią a praktyką, pokazując, jak przekształcić fascynujące demonstracje w działające produkty biznesowe. Stanowi doskonałe uzupełnienie publikacji Sebastiana Raschki **Stwórz własne AI. Jak od podstaw zbudować duży model językowy**, skupionej na budowaniu i zrozumieniu LLM od podstaw — rozszerza tę wiedzę o praktyczne zastosowania w produkcji, w tym integrację, efektywne kosztowo trenowanie modeli i ocenę ich jakości.

Autorzy prowadzą czytelnika przez kompletny proces — od wyboru odpowiedniego modelu bazowego, przez przygotowanie danych treningowych i techniki dostrajania, aż po wdrożenie w klastrze Kubernetes. Prezentują przy tym trzy praktyczne projekty: budowę własnego modelu LLM od podstaw, stworzenie rozszerzenia AI dla VS Code i wdrożenie modelu na Raspberry Pi. Szczególny nacisk kładą na aspekty produkcyjne — zarządzanie kosztami, bezpieczeństwo, skalowanie i monitorowanie systemów opartych na LLM.

Christopher Brousseau — specjalista do spraw uczenia maszynowego w JPMorganChase, ekspert w dziedzinie lingwistyki i przetwarzania języka naturalnego.

Matthew Sharp — doświadczony inżynier i lider technologiczny MLOps, specjalizuje się we wdrażaniu i skalowaniu modeli ML w środowiskach produkcyjnych.

Książka opisuje wszystkie kluczowe aspekty budowania i wdrażania dużych modeli językowych. Obejmuje szczegółowe i fascynujące obszary, pomijane w większości innych publikacji.

Andrew Carr
Cartwheel

Lektura obowiązkowa dla każdego, kto chce wykorzystać potencjał dużych modeli językowych w środowiskach produkcyjnych.

Jepson Taylor
VEOX Inc.

Wyjątkowy przewodnik, który upraszcza proces budowy i wdrażania złożonych dużych modeli językowych.

Arunkumar Gopalan
Microsoft

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 978-83-289-3304-0

Cena: 149,00 zł

MANNING