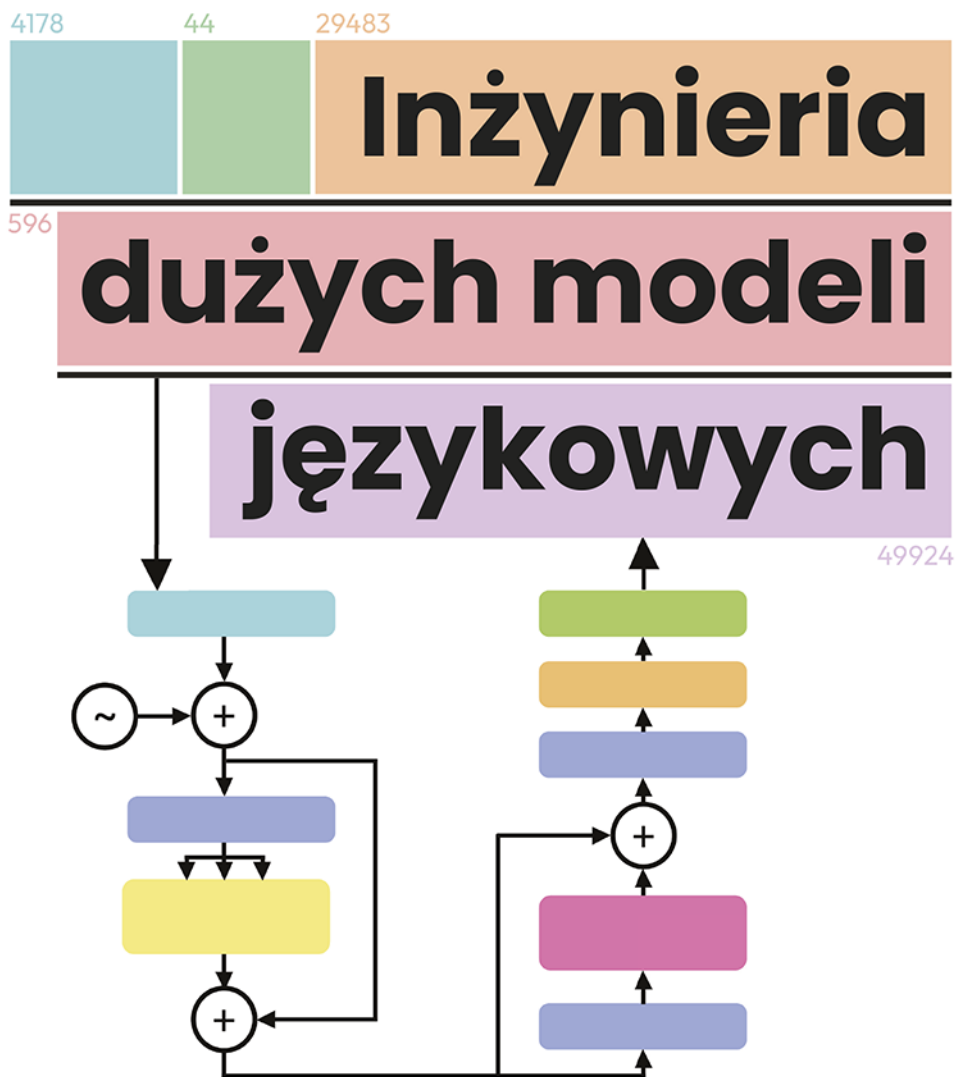




Podręcznik projektowania, trenowania i wdrażania LLM

Przedmowa

Julien Chaumond
współzałożyciel i CTO
Hugging Face Hub

Hamza Tahir
współzałożyciel
i CTO ZenML

Antonio Gulli
starszy dyrektor
w Google



Tytuł oryginału: LLM Engineer's Handbook: Master the art of engineering large language models from concept to production

Tłumaczenie: Robert Górczyński

ISBN: 978-83-289-2530-4

Copyright © Packt Publishing 2024. First published in the English language under the title 'LLM Engineer's Handbook' – (9781836200079).

Polish edition copyright © 2025 by Helion S.A.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

helion.pl/user/opinie/indumo

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.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści |

O autorach	11
O korektorze merytorycznym	12
Przedmowa	13
Wprowadzenie	17
ROZDZIAŁ 1	
Architektura i koncepcja projektu LLM Twin	23
Koncepcja kryjąca się za aplikacją LLM Twin	24
Czym jest projekt LLM Twin?	24
Dlaczego sensowne jest tworzenie projektu LLM Twin?	25
Dlaczego w omawianym celu nie można użyć ChatGPT (lub podobnego chatbota)?	27
Planowanie produktu o minimalnej niezbędnej funkcjonalności dla projektu LLM Twin	28
Czym jest produkt o minimalnej niezbędnej funkcjonalności?	29
Zdefiniowanie produktu o minimalnej niezbędnej funkcjonalności w projekcie LLM Twin	29
Budowanie systemu uczenia maszynowego z wykorzystaniem potoków cech, trenowania i wnioskowania	31
Aspekty, jakie należy uwzględnić podczas budowania systemów uczenia maszynowego	31
Problem z poprzednimi rozwiązaniami	32
Rozwiązanie — potoki uczenia maszynowego dla systemów uczenia maszynowego	35
Zalety architektury FTI	37
Opracowanie architektury systemu dla projektu LLM Twin	39
Szczegóły techniczne dotyczące infrastruktury projektu LLM Twin	39
Jak opracować architekturę LLM Twin za pomocą projektu opartego na potoku FTI?	40
Kilka uwag końcowych na temat projektu FTI i architektury LLM Twin ...	46
Podsumowanie	47
Źródła	47

ROZDZIAŁ 2

Narzędzia i ich instalacja	49
Ekosystem Pythona i przygotowanie projektu	50
Poetry — menedżer zależności i środowisk wirtualnych	51
Poe the Poet — narzędzie do wykonywania zadań	53
Narzędzia MLOps i LLMOps	54
Hugging Face — rejestr modelu	54
ZenML — oprogramowanie koordynujące, artefakty i metadane	56
Comet — oprogramowanie do śledzenia eksperymentu	66
Opik — monitorowanie promptu	68
Bazy danych do przechowywania danych niestrukturyzowanych i wektorowych	68
MongoDB — baza danych typu NoSQL	69
Qdrant — wektorowa baza danych	69
Przygotowanie do użycia chmury AWS	70
Utworzenie konta AWS i klucza dostępu oraz przygotowanie narzędzia powłoki do pracy z usługą AWS	70
SageMaker — obliczenia związane z trenowaniem i wnioskowaniem	72
Dlaczego SageMaker?	72
Podsumowanie	74
Źródła	74

ROZDZIAŁ 3

Inżynieria danych	76
Opracowanie potoku pobierania danych do projektu LLM Twin	77
Implementacja potoku pobierania danych do projektu LLM Twin	81
Potok ZenML i kroki	81
Dyspozytor — jak zainicjalizować odpowiedni crawler?	85
Crawlers	87
Dokumenty hurtowni danych typu NoSQL	96
Umieszczanie nieprzetworzonych danych bezpośrednio w hurtowni danych	104
Rozwiązywanie problemów	108
Podsumowanie	109
Źródła	109

ROZDZIAŁ 4

Potok wykorzystujący technikę RAG	111
Wyjaśnienie techniki RAG	112
Dlaczego warto używać techniki RAG?	112
Zwykły framework systemu RAG	114
Czym są osadzenia?	118
Więcej informacji na temat wektorowych baz danych	125
Ogólne omówienie zaawansowanej techniki RAG	127
Przed pobraniem danych	129
Pobieranie danych	132
Po pobraniu danych	134
Prezentacja architektury techniki RAG wykorzystanej w projekcie LLM Twin ...	136
Problem, który chcemy rozwiązać	136
Magazyn danych cech	137
Skąd pochodzą nieprzetworzone dane?	138
Opracowanie architektury potoku cech techniki RAG	138
Implementacja potoku techniki RAG w projekcie LLM Twin	148
Klasa Settings	148
Potok ZenML i kroki	149
Encje dziedziny Pydantic	157
Warstwa dyspozytora	164
Procedury obsługi	167
Podsumowanie	175
Źródła	176

ROZDZIAŁ 5

Nadzorowane dostrajanie modelu	177
Tworzenie wysokiej jakości zbioru danych instrukcji	178
Ogólny framework rozwiązania	178
Gromadzenie danych	181
Filtrowanie oparte na regułach	182
Eliminacja duplikatów	183
Dekontaminacja danych	185
Ocena jakości danych	186
Eksploracja danych	188
Generowanie danych	190
Uzupełnienie danych	192
Tworzenie własnego zbioru danych instrukcji	194

Nadzorowane dostrajanie modelu i związanych z nim technik	202
Kiedy należy dostrajać model?	203
Formaty zbiorów danych instrukcji	204
Szablony czatu	205
Techniki Parameter-Efficient Fine-Tuning (PEFT)	206
Trenowanie parametrów	212
Dostrajanie w praktyce	215
Podsumowanie	222
Źródła	222

ROZDZIAŁ 6

Dostrajanie modelu z uwzględnieniem preferencji użytkowników 224

Poznanie zbiorów danych preferencji	225
Dane preferencji	225
Generowanie danych i ich ocena	228
Samodzielne tworzenie zbioru danych preferencji	233
Uwzględnienie preferencji	239
Uczenie przez wzmocnienie na podstawie opinii użytkowników	239
Bezpośrednia optymalizacja preferencji	241
Implementacja bezpośredniej optymalizacji preferencji	244
Podsumowanie	250
Źródła	251

ROZDZIAŁ 7

Ocena dużych modeli językowych 252

Ocena modelu	252
Porównanie oceny uczenia maszynowego i oceny dużego modelu językowego	253
Ocena dużego modelu językowego ogólnego przeznaczenia	254
Ocena dużego modelu językowego związanego z dziedziną	256
Ocena dużego modelu językowego związanego z zadaniem	258
Ocena systemu RAG	261
Ragas	262
ARES	264
Ocena modelu TwinLlama-3.1-8B	265
Generowanie odpowiedzi	266
Ocena odpowiedzi	268
Analiza wyników	271
Podsumowanie	275
Źródła	276

ROZDZIAŁ 8

Optymalizacja wnioskowania	277
Strategie optymalizacji modelu	278
Bufor KV	279
Przetwarzanie ciągłymi partiami	281
Dekodowanie spekulatywne	282
Zoptymalizowane mechanizmy uwagi	284
Równoległość modelu	285
Równoległość danych	286
Równoległość potoku	287
Równoległość tensora	288
Łączenie różnych technik	289
Kwantyzacja modelu	290
Wprowadzenie do kwantyzacji	291
Kwantyzacja za pomocą GGUF i llama.cpp	295
Kwantyzacja za pomocą GPTQ i EXL2	297
Inne techniki kwantyzacji	298
Podsumowanie	299
Źródła	300

ROZDZIAŁ 9

Potok wnioskowania wykorzystujący technikę RAG	301
Potok wnioskowania RAG w modelu Twin	302
Zaawansowane techniki RAG w modelu Twin	304
Zaawansowane techniki optymalizacji etapu przed pobieraniem danych w systemie RAG — rozbudowa zapytania i samozapytanie	307
Zaawansowane techniki optymalizacji etapu pobierania danych w systemie RAG — filtrowane wyszukiwanie wektorowe	314
Zaawansowane techniki optymalizacji etapu po pobieraniu danych w systemie RAG — ponowne przygotowanie rankingu	315
Implementacja potoku wnioskowania RAG w modelu Twin	319
Implementacja modułu pobierania danych	319
Połączenie wszystkiego w całość w potoku wnioskowania w systemie RAG	325
Podsumowanie	330
Źródła	331

ROZDZIAŁ 10

Wdrożenie potoku wnioskowania	332
Kryteria wyboru rodzaju wdrożenia	333
Przepustowość i opóźnienie	333
Dane	334
Infrastruktura	334
Różne typy wdrożeń potoku wnioskowania	336
Wnioskowanie online w czasie rzeczywistym	337
Wnioskowanie asynchroniczne	338
Przekształcanie partiami w trybie offline	339
Architektura monolityczna i architektura mikrousług w infrastrukturze udostępniania modelu	340
Architektura monolityczna	341
Architektura mikrousług	341
Wybór między architekturą monolityczną i architekturą mikrousług	343
Strategia wdrażania potoku wnioskowania projektu LLM Twin	344
Potok wnioskowania i potok trenowania	347
Wdrażanie usługi LLM Twin	348
Implementowanie mikrousługi dużego modelu językowego za pomocą AWS SageMakera	349
Budowanie mikrousługi biznesowej za pomocą FastAPI	363
Automatyczne skalowanie możliwości w celu obsługi nagłego wzrostu poziomu użycia usługi	366
Rejestrowanie skalowanego celu	368
Tworzenie polityki skalowania	369
Wartości minimalna i maksymalna podczas skalowania	370
Okres oczekiwania	370
Podsumowanie	371
Źródła	372

ROZDZIAŁ 11

MLOps i LLMOps	373
Ścieżka prowadząca do LLMOps — korzenie w podejściach DevOps i MLOps	374
DevOps	374
MLOps	377
LLMOps	381
Wdrożenie w chmurze potoków projektu LLM Twin	387
Elementy infrastruktury	387
Konfiguracja bazy danych MongoDB	389
Konfiguracja bazy danych Qdrant	390
Konfiguracja chmury ZenML	392

Dodanie LLMOps do projektu LLM Twin	402
Przepływ pracy w potoku CI/CD projektu LLM Twin	403
GitHub Actions	405
Potok CI	407
Potok CD	410
Testowanie potoku CI/CD	412
Potok CT	414
Monitorowanie promptu	419
Ostrzeganie	423
Podsumowanie	424
Źródła	425

DODATEK A

Reguły MLOps	426
---------------------------	------------

Nadzorowane dostrajanie modelu

Rozdział

5

Nadzorowane dostrajanie modelu (ang. *supervised fine-tuning*, SFT) to niezwykle istotny krok podczas przygotowywania dużych modeli językowych do użycia w rzeczywistych aplikacjach. Przeprowadzane po etapie początkowego wstępnego trenowania, w trakcie którego duży model językowy uczy się prognozować następny token w sekwencji, nadzorowane dostrajanie modelu udoskonala możliwości modelu za pomocą starannie przygotowanych par instrukcji i udzielonych na nie odpowiedzi. Ten proces służy dwóm najważniejszym celom. Pierwszy polega na tym, aby model nauczył się rozumieć konkretny format czatu i się do niego dostosować, co w praktyce oznacza, że model staje się agentem, z którym można prowadzić rozmowę. Drugim celem jest umożliwić modelowi dostosowywanie jego szerokiej bazowej wiedzy do wykonywania konkretnych zadań w wyspecjalizowanych dziedzinach.

Waga nadzorowanego dostrajania modelu kryje się w możliwości wypełnienia luki między rozumieniem ogólnego języka, którego używają ludzie, i praktycznym jego wykorzystaniem. Dzięki przedstawieniu modelowi przykładów oczekiwanych wzorców danych wejściowych i wyjściowych nadzorowane dostrajanie kształtuje sposób działania dużego modelu językowego, aby pozostawał zgodny z określonymi celami, niezależnie od tego, czy obejmują one wykonanie danego zadania (np. wygenerowanie podsumowania lub przetłumaczenia tekstu), czy podzielenie się wiedzą z danej dziedziny (np. medycyny lub prawa). Tego rodzaju podejście nie tylko poprawia wydajność działania modelu w określonych dziedzinach, ale również zwiększa jego możliwości w zakresie stosowania się do poleceń i generowania znacznie dokładniejszych i bardziej spójnych odpowiedzi.

W tym rozdziale skoncentrujemy się na następujących zagadnieniach:

- tworzenie wysokiej jakości zbioru danych instrukcji;
- techniki nadzorowanego dostrajania modelu;
- praktyczna implementacja nadzorowanego dostrajania modelu.

Podczas lektury tego rozdziału nauczysz się samodzielnie tworzyć zbiory danych i efektywnie używać ich do dostrajania dużych modeli językowych.

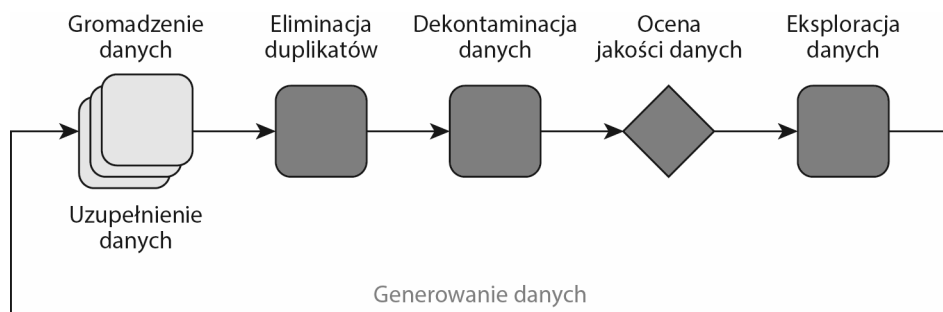
Uwaga

Wszystkie przykładowe fragmenty kodu znajdują się w repozytorium GitHub pod adresem <https://github.com/PacktPublishing/LLM-Engineers-Handbook>.

Tworzenie wysokiej jakości zbioru danych instrukcji

W większości przypadków utworzenie zbioru danych instrukcji okazuje się najtrudniejszym etapem w procesie dostrajania. To wynika z wielu czynników. Większość przypadków użycia można powiązać z nieprzetworzonym tekstem, choć bardzo rzadko znajduje się naturalne pary instrukcji i odpowiedzi. Ten nieprzetworzony tekst musi być przekształcony na format, który będzie zawierał zarówno instrukcje, jak i odpowiedzi. Co więcej, bardzo istotna jest tutaj jakość danych. Z tego powodu mnóstwo czasu poświęca się na ręczne sprawdzenie i zweryfikowanie poszczególnych próbek. Taki dokładny proces weryfikacji danych pomaga zagwarantować, że zbiór danych jest poprawny i użyteczny na potrzeby związane z trenowaniem modelu.

W tym podrozdziale przedstawimy ogólny framework, który służy do tworzenia własnych zbiorów danych, niezależnie od ostatecznego przypadku użycia. Następnie wykorzystamy dane pobrane za pomocą potoku omówionego w rozdziale 3. i przekształcimy je na postać zbioru danych instrukcji. Poszczególne etapy potoku generowania danych zostały schematycznie pokazane na rysunku 5.1.



Rysunek 5.1. Ogólny schemat potoku danych omawianego w tym rozdziale

Ogólny framework rozwiązania

Zbiór danych instrukcji jest definiowany w postaci par pytań i odpowiedzi na nie. Te pytania to dane wejściowe modelu używane jako kontekst dla jego dostrajania. Natomiast odpowiedzi to oczekiwane od modelu dane wyjściowe. Podczas operacji dostrajania można wybierać między trenowaniem modelu na podstawie pytań i odpowiedzi bądź jedynie odpowiedzi. Pary składające się z pytania i odpowiedzi są tworzone z użyciem określonego szablonu. Niektóre szablony instrukcji, np. Alpaca, zawierają dodatkowe pola, takie jak `inputs` i `system`. Oba mogą być uznane za podpola dla pola `instruction`. W takim przypadku pole `inputs` zawiera dane, których model potrzebuje, by wykonać polecenie, pole `system` zaś to metaprompt wskazujący ogólny sposób działania modelu. W przedstawionym w tabeli 5.1 przykładzie pochodzącym ze zbioru danych SlimOrca mamy wszystkie wymienione pola.

Tabela 5.1. Przykład instrukcji pochodzącej ze zbioru danych Open-Orca/SlimOrca

System

Jesteś pomocnym asystentem, który wszystko dokładnie wyjaśnia. Odpowiadaj tak, jakbyś rozmawiał z pięcioletkiem.

Instrukcja

Pojęcia: budynek, sklep, miasto

Utwórz zdanie, które zawiera wszystkie wymienione słowa.

Dane wyjściowe

W naszym małym mieście znajduje się duży budynek mieszczący sklep, w którym klienci kupują ulubione zabawki i słodycze.

Przykład zamieszczony w tabeli 5.1 pokazuje, jak pole „systemu” zostało użyte do zdefiniowania określonego sposobu działania modelu jako pomocnego i wyjaśniającego wszystko dokładnie asystenta, którego odpowiedzi zostały dopasowane do poziomu pięcioletka. Z kolei pole „instrukcji” dostarcza niezbędne dane (pojęcia) i zadanie (utworzenie zdania). Natomiast pole „danych wyjściowych” zawiera oczekiwaną odpowiedź, która wprawdzie nie jest jedyną możliwą, ale stanowi wysokiej jakości odpowiedź na pytanie.

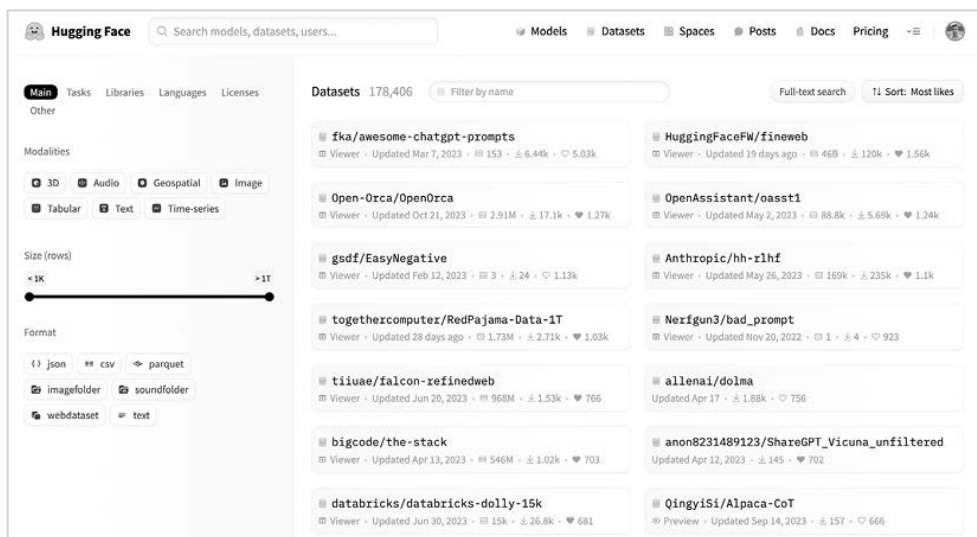
Aby opracować zbiór danych, trzeba zgromadzić dane, które będą reprezentatywne dla sposobu działania modelu. Po zebraniu odpowiedniej liczby próbek celem jest ich przefiltrowanie i pozostawienie jedynie wysokiej jakości danych. W tym kontekście wysokiej jakości dane można opisać za pomocą trzech najważniejszych aspektów:

- **Dokładność.** Odwołuje się do rzeczywistej poprawności i trafności próbek. W kontekście zbiorów danych instrukcji oznacza to, że udzielone odpowiedzi muszą być nie tylko poprawne, ale również trafne z perspektywy pytań. Osiągnięcie wysokiej dokładności ma istotne znaczenie dla procesu trenowania modeli, które będą w stanie dostarczać rzetelne i wiarygodne informacje.
- **Różnorodność.** Wysokiej jakości zbiór danych powinien obejmować szeroką gamę przypadków użycia i mieć zastosowanie podczas przetwarzania potencjalnych zapytań i wykonywania zadań zleczanych dużemu modelowi językowemu. Różnorodność powinna obejmować tematykę, kontekst, objętość treści i jej styl. Dzięki próbkowaniu danych w reprezentatywny sposób umożliwiamy modelowi wypracowanie niezawodnego stosowania się do poleceń.
- **Złożoność.** Trywialne lub przesadnie uproszczone próbki mają niewielki wpływ na poprawę możliwości dużego modelu językowego. Dlatego zbiory danych powinny zawierać skomplikowane i składające się z wielu kroków wnioskowania rozwiązania problemów i trudnych zadań, które będą przesuwały granice możliwości modelu. Taka złożoność pomaga w opracowaniu modeli, które będą radzić sobie z rzeczywistymi i złożonymi problemami.

W kolejnych punktach przedstawimy techniki filtrowania i oceny próbek instrukcji zgodnie z wymienionymi kryteriami.

Ilość danych

Serwis Hugging Face Hub (rysunek 5.2) zawiera wiele zbiorów danych instrukcji ogólnego przeznaczenia albo opracowanych z myślą o konkretnych zadaniach lub dziedzinach. Podczas pracy nad nowym przypadkiem użycia dobrym rozwiązaniem może być sięgnięcie po otwartoźródłowe zbiory danych i wykorzystanie ich do dostrojenia modelu. Jest to szczególnie ważne w przypadku zbyt małej liczby próbek (np. poniżej 1000), co będzie wymagało uzupełnienia ich danymi wysokiej jakości.



Rysunek 5.2. Najpopularniejsze zbiory danych w serwisie Hugging Face Hub

Określenie idealnej liczby próbek jest trudnym zadaniem, ponieważ zarówno jakość danych, jak i wielkość modelu mogą mieć na to ogromny wpływ. W przypadku dużych modeli (np. mających około 70 miliardów parametrów) ta liczba może wynosić zaledwie 1000 wysokiej jakości próbek (zapoznaj się z opracowaniem LIMA, o którym wspomnieliśmy w podrozdziale „Źródła”). Sytuacja wygląda zupełnie inaczej dla mniejszych modeli (np. mających około 7 miliardów parametrów), ponieważ wymagają one więcej próbek, aby poznać poprawny szablon czatu. Jednak w obu tych przypadkach jakość danych ma istotne znaczenie, a większa liczba próbek zawsze będzie mile widziana.

Aby podać jeszcze inne liczby próbek, warto zapoznać się z dostrojonymi modelami, które zostały udostępnione przez firmy i społeczność otwartoźródłową. Rozróżniamy dwa rodzaje operacji dostrojenia: ogólnego przeznaczenia, której celem jest odtworzenie możliwości modeli takich jak ChatGPT, oraz związaną z zadaniem bądź dziedziną, mającą na celu optymalizację wydajności działania modelu w konkretnej aplikacji.

Modele ogólnego przeznaczenia obsługują więcej tematów, co będzie wymagało dodatkowych próbek. Wśród firm można zauważyć stosowanie szerokiego przedziału wartości. Na przykład modele Yi z 01-ai zostały dostrojone za pomocą mniej niż 10 000 próbek. Na drugim końcu spektrum znajduje się Meta i jej model Llama 3, do którego dostrojenia użyto 10 milionów próbek (obejmuje to również uwzględnienie preferencji).

W społeczności otwartoźródłowej modele takie jak OpenHermes i Dolphin dostroją do pomocą miliona próbek. Dla zapewnienia jakości tych operacji dostrojenia zalecamy, aby zbiór danych składał się z przynajmniej miliona próbek, gdyż pozwoli to uzyskać model ogólnego przeznaczenia o dobrej jakości. Z kolei modele dostrajane do konkretnych zadań będą wymagały mniej próbek. W tym przypadku należy odróżnić modele przeznaczone do określonych zadań od modeli ściśle związanych z daną dziedziną.

Modele związane z zadaniem lub dziedziną reprezentują dwa odmienne podejścia do dostrajania dużych modeli językowych. Te pierwsze mają doskonale się sprawdzać w określonej funkcji, na przykład tłumaczeniu, generowaniu podsumowania bądź analizie nastrojów. W takim przypadku korzystne jest podejście polegające na uczeniu, które koncentruje się na pojedynczym zadaniu. Dzięki temu można uzyskać efektywność działania, nawet w przypadku mniejszych modeli (zwykle mających poniżej 8 miliardów parametrów). Dane niezbędne do dostrojenia modelu przeznaczonego do konkretnego zadania są przeważnie łatwiejsze do zarządzania, a ich ilość waha się w przedziale od 100 do 100 000. Z tego powodu dostrajanie pod kątem konkretnego zadania okazuje się atrakcyjnym rozwiązaniem w przypadku wielu aplikacji, które mają ograniczoną ilość zasobów.

Modele związane z dziedziną natomiast mają na celu dostrojenie dużego modelu językowego z użyciem wiedzy specjalistycznej, a także nauczanie słownictwa i wzorców językowych określonej dziedziny. Takie modele okazują się cenne na obszarach takich jak medycyna, prawo, finanse, e-commerce, inżynieria oraz hotelarstwo. Wymagania dotyczące danych, które są używane do dostrajania takich modeli, mogą być bardzo różne, zależnie od złożoności i obszerności dziedziny. W niektórych dziedzinach, na przykład w medycynie i prawie, modele mogą wymagać równie dużo danych jak modele ogólnego przeznaczenia, co wynika z rozległości tych dziedzin. Z kolei inne, na przykład e-commerce i hotelarstwo, mogą potrzebować mniej próbek, czyli podobnie jak ma to miejsce w przypadku dostrajania pod kątem określonego zadania.

Kluczowymi czynnikami, które wpływają na ilość danych niezbędnych podczas dostrajania modeli związanych z określoną dziedziną, są „wielkość” dziedziny (np. zakres wiedzy specjalistycznej i słownictwo) oraz reprezentacja dziedziny w danych wykorzystanych podczas wstępnego trenowania modelu. Dziedziny, które są dobrze reprezentowane w pierwotnych danych użytych do trenowania modelu, mogą wymagać mniej danych w procesie dostrajania. Natomiast dziedziny bardziej specjalistyczne lub niewystarczająco reprezentowane w zbiorze uczącym mogą wymagać obszerniejszych zbiorów danych podczas dostrajania. Nawet w przypadku modeli otwartoźródłowych wiele zbiorów danych używanych do wstępnego trenowania jest własnościami, co będzie wymagało zgadywania ich proporcji (np. 30% kodu źródłowego lub 20% wzorów matematycznych).

Gromadzenie danych

W zakresie pozyskiwania danych na potrzeby dostrajania modelu stosowane podejścia różnią się dla modeli związanych z zadaniem i modeli związanych z dziedziną. W przypadku tych pierwszych pozyskiwanie danych często obejmuje zbieranie przykładów oczekiwanych żądań, które pochodzą z istniejących zbiorów danych bądź z nowo

utworzonych. To może oznaczać zbieranie par tekstu pierwotnego i jego podsumowania (w przypadku modelu generującego podsumowania) albo pobieranie zdań w różnych językach (w przypadku modelu tłumaczącego tekst).

Operacja pozyskiwania danych dla modelu związanego z dziedziną może okazać się znacznie trudniejszym zadaniem. Często wymaga pracy z ekspertami z danej dziedziny, aby zebrać i zweryfikować artykuły, prace naukowe, dokumenty techniczne oraz inne teksty ściśle związane z daną dziedziną. W niektórych sytuacjach to może oznaczać nawet współpracę z organizacjami lub instytucjami, które mają dostęp do ogromnych repozytoriów specjalistycznych informacji. Jakość i trafność tych danych są niezwykle istotne, ponieważ bezpośrednio wpływają na możliwości modelu w zakresie rozumienia i generowania treści z określonej dziedziny.

W tym miejscu trzeba wspomnieć o pojawieniu się strategii nazywanej *few-shot prompting* jako alternatywy dla dostrajania, zwłaszcza w przypadku zastosowań związanych z zadaniem. Takie podejście wykorzystuje możliwości ogromnych modeli poprzez dostarczenie w prompcie kilku przykładów oczekiwanego zadania. Wprawdzie ta strategia nie jest zamiennikiem dostrajania modelu w każdej sytuacji (np. gdy chcesz poznać nową dziedzinę), ale może być efektywnym sposobem przystosowania modeli do nowych zadań, bez konieczności przeprowadzania rozległego, dodatkowego trenowania.

W praktyce granica między modelami związanymi z zadaniem i modelami związanymi z dziedziną może się czasem zacierać. Na przykład dostrajanie modelu używanego do diagnoz medycznych może być uznawane za związane z zadaniem (koncentracja na diagnozach) lub dziedziną (specjalizacja z określonej dziedziny medycyny). Kluczem jest zrozumienie podstawowego celu dostrajania i odpowiednie dostosowanie tego procesu do własnych potrzeb.

Na tym etapie procesu powinniśmy mieć kolekcję zbiorów danych, które są dostosowane do konkretnego przypadku użycia. Następny krok polega na dopracowaniu jakości próbek za pomocą filtrowania opartego na regułach, eliminacji powielonych danych, dekontaminacji danych i oceny jakości danych.

Filtrowanie oparte na regułach

To systematyczne podejście do kontroli jakości danych, które opiera się na jasno zdefiniowanych regułach oceny i filtrowania próbek danych. Te reguły są zwykle opracowywane w celu wyeliminowania najczęściej pojawiających się problemów z jakością oraz mieszczą się w zakresie od prostych operacji sprawdzenia do znacznie bardziej skomplikowanych operacji logicznych. Podstawowym celem filtrowania opartego na regułach jest utrzymanie wysokich standardów jakości danych i usunięcie próbek, które nie spełniają określonych kryteriów.

Filtrowanie według wielkości to prosta i zarazem efektywna technika filtrowania. Obejmuje zdefiniowanie wartości progowych dla akceptowanej długości odpowiedzi w zbiorze danych. Wyjątkowo krótkie odpowiedzi przeważnie nie mają wystarczającej ilości informacji, natomiast wyjątkowo długie mogą zawierać nieistotną lub zbędną treść. Należy pamiętać, że waga odpowiednich wartości progowych może być różna w zależności od określonego zadania i dziedziny. Na przykład zbiór danych przeznaczony

do wygenerowania zwięzłych podsumowań może mieć mniejszą maksymalną wartość progową niż zbiór przeznaczony do generowania szczegółowych objaśnień.

Wyłączanie określonych słów kluczowych to technika filtrowania na podstawie reguł, w której nacisk kładzie się na próbki, a nie na ich strukturę. Obejmuje utworzenie listy słów kluczowych bądź fraz powiązanych z treścią nieodpowiednią lub o niskiej jakości, a następnie odfiltrowanie wszelkich próbek zawierających te wyrażenia. Lista słów kluczowych może obejmować oczywiste treści o niskiej jakości, np. wulgaryzmy i spam, a także ściśle związane z daną dziedziną słowa, które mogą wskazywać na brak związku z tematem. Na przykład w zbiorze danych przeznaczonym dla profesjonalnego asystenta pisarza można wykluczyć próbki zawierające określenia slangowe bądź potoczne, które nie pasują do wybranego tonu i stylu.

Sprawdzanie formatu jest zalecane dla zbiorów danych, które zawierają dane strukturalne bądź spełniają określone wymogi w zakresie formatowania. Ta technika gwarantuje, że wszystkie próbki są zgodne z oczekiwanym formatem, co z kolei ułatwia zachowanie spójności i późniejsze przetwarzanie danych. Sprawdzanie formatu może okazać się szczególnie ważne w przypadku zbiorów danych, które zawierają próbki kodu źródłowego, struktury JSON oraz inny sformatowany tekst. Na przykład zbiór danych instrukcji programistycznych może implementować reguły weryfikujące, czy próbki kodu są syntaktycznie poprawne oraz spełniają określone reguły stylu.

Filtrowanie oparte na regułach zapewnia duże korzyści podczas przygotowywania zbiorów danych instrukcji. Szybkość działania i efektywność tej strategii umożliwiają jej stosowanie dla ogromnych ilości danych, co oznacza, że takie rozwiązanie jest wysoce skalowalne. Spójność w zakresie stosowania reguł pozwala w jednakowy sposób przetwarzać dane, a tym samym ograniczyć ryzyko błędu ludzkiego oraz ewentualną tendencyjność. Co więcej, jasno zdefiniowane kryteria filtrowania gwarantują przejrzystość i interpretowalność, a także ułatwiają zrozumienie, audyt i dostosowanie danych. Możliwość automatyzacji filtrowania opartego na regułach zmniejsza potrzebę ręcznej interwencji oraz pozwala zastosować nieustanne monitorowanie jakości danych.

Jednak filtrowanie oparte na regułach ma również ograniczenia, które należy wziąć pod uwagę. Predefiniowane reguły mogą nie obejmować szczegółów wymaganych do przechwycenia pełnej złożoności języka i kontekstu, co może prowadzić do usunięcia poprawnych, choć nietypowych próbek. Typowo binarna natura próbek (zaliczony/niezaliczony) nie zawsze musi pozostawać w zgodzie z naturą języka i jakością instrukcji. Ponadto wzorce danych i standardy jakości ewoluują, więc reguły wymagają regularnych przeglądów i uaktualnień, aby zawsze były efektywne. Istnieje również ryzyko, że słabo opracowane reguły mogą przypadkowo wprowadzić lub wzmocnić błędy w zbiorze danych.

Eliminacja duplikatów

Zróznicowanie zbioru danych jest podstawą procesu trenowania modelu, który potrafi uogólniać wyniki dla zupełnie nowych danych. Gdy zbiór danych zawiera duplikaty lub prawie duplikaty, może to prowadzić do wielu problemów:

- **Przetrenowanie.** Model może zapamiętywać konkretne przykłady zamiast uczyć się ogólnych wzorców.
- **Tendencyjność związana z wydajnością działania.** Nadmiernie reprezentowane punkty danych mogą wypaczyć wydajność działania w kierunku określonego typu danych wejściowych.
- **Niewystarczające trenowanie.** Nadmierne dane mogą wydłużyć czas trenowania, nie dostarczając przy tym dodatkowo żadnych cennych informacji.
- **Nadmuchane wskaźniki oceny.** Powielanie danych w zbiorach testowych może prowadzić do nazbyt optymistycznych ocen wydajności działania.

W celu wyeliminowania duplikatów można przeprowadzić operację dokładnego lub przybliżonego pozbycia się powielonych danych. W przypadku **dokładnej eliminacji duplikatów** następuje usunięcie identycznych próbek za pomocą prostego procesu, który obejmuje normalizację danych, generowanie skrótów oraz usuwanie duplikatów. Normalizacja danych standaryzuje format elementów danych, np. przeprowadza konwersję tekstu na postać zapisaną małymi literami. Generowanie skrótów prowadzi do wygenerowania unikatowych wartości skrótów dla poszczególnych elementów za pomocą algorytmów takich jak MD5 lub SHA-256. Te wartości skrótów są porównywane w celu znalezienia duplikatów, które następnie są usuwane. W efekcie pozostaje tylko jeden egzemplarz danego elementu danych. Wprawdzie ta strategia jest efektywna dla identycznych elementów, ale nie radzi sobie z wykrywaniem prawie duplikatów bądź semantycznie podobnej treści — w tych przypadkach konieczne jest zastosowanie bardziej zaawansowanych technik.

Najpopularniejsze podejście w zakresie **przybliżonej eliminacji duplikatów** to technika MinHash. W porównaniu do innych jest wysoce skuteczna, a jednocześnie znacznie prostsza. MinHash działa poprzez wygenerowanie zwężonych reprezentacji, inaczej sygnatur, dla poszczególnych elementów danych. Te sygnatury działają jako odciski palca przechwytyjące istotę danych, a zarazem znacznie zmniejszają ich wymiarowość. W praktyce MinHash konwertuje elementy danych (takie jak dokumenty tekstowe) na zestawy, stosuje wiele funkcji skrótów w tych zbiorach, a następnie wybiera minimalne wartości skrótu w celu przygotowania wektorów sygnatur. Te sygnatury mogą być porównywane za pomocą technik sprawdzania podobieństwa (np. podobieństwo Jaccarda), aby znaleźć niemalże identyczne duplikaty.

Poza omówionymi tutaj technikami eliminacji duplikatów **podobieństwo semantyczne** stosuje inne podejście i koncentruje się na znaczeniu tekstu. Ta metoda obejmuje konwersję słów lub całych próbek na reprezentacje wektorowe za pomocą technik przetwarzania języka naturalnego. Modele osadzania słów, np. Word2Vec, GloVe i FastText, konwertują poszczególne słowa na gęste wektory i przechwytyują semantyczne podobieństwo.

W przypadku reprezentacji bardziej związanych z kontekstem modele językowe takie jak BERT, transformery zdań bądź enkodery krzyżowe mogą generować osadzenia dla całych zdań lub dokumentów. Po otrzymaniu tych wektorów eliminacja duplikatów może zostać przeprowadzona poprzez porównanie podobieństwa między wektorami. Do często stosowanych metod zaliczamy podobieństwo cosinusa i odległość Euklidesową. Próbkę z wysokim wynikiem podobieństwa ponad pewną wartość progową mogą być uznane za duplikaty. W przypadku ogromnych zbiorów danych techniki klastrowania

można zastosować dla grup podobnych wektorów. Metody takie jak algorytm k-średnich, DBSCAN lub klastrowanie hierarchiczne umożliwiają efektywne organizowanie przestrzeni wektorowej oraz identyfikowanie klastrów, które przedstawiają semantycznie podobną treść. W ramach poszczególnych klastrów można pobrać reprezentatywną próbkę, pozostałe zaś oznaczyć jako duplikaty.

Dekontaminacja danych

Dekontaminacja danych to proces, który ma zagwarantować, że zbiór danych używany do trenowania modelu nie zawiera próbek identycznych lub bardzo podobnych do próbek wykorzystywanych w trakcie oceny bądź testu modelu. Ten krok ma istotne znaczenie dla jakości oceny modelu oraz zapobieżenia przetrenowaniu lub zapamiętywaniu danych testowych.

W trakcie dekontaminacji danych są stosowane techniki znane z procesów eliminacji duplikatów. Dokładne dopasowanie może posłużyć do usunięcia wszelkich próbek, które są identyczne jak znajdujące się w zbiorach używanych do oceny modelu. Tę operację można przeprowadzić za pomocą funkcji skrótu lub bezpośredniego porównywania ciągów tekstowych. Następnie można skorzystać z metod wykrywania prawie duplikatów, aby zidentyfikować i usunąć próbki używane podczas trenowania modelu, które są podobne do próbek stosowanych podczas oceny modelu, nawet jeśli nie są identyczne. To często będzie oznaczało używanie technik takich jak MinHash lub obliczanie prawdopodobieństwa na podstawie n-gramów lub osadzeń.

Uwaga

Prostym sposobem na przeprowadzenie dekontaminacji danych jest dodanie zbioru używanego do oceny modelu do zbioru danych instrukcji na etapie eliminowania duplikatów. W takim przypadku chcemy zagwarantować, że ze zbioru danych instrukcji będą usuwane jedynie te próbki, które mogą być zaimplementowane na różne sposoby (np. odfiltrowanie pierwszego duplikatu, zarejestrowanie indeksów próbek używanych do oceny modelu). W idealnej sytuacji zbiory danych przeznaczone do oceny modelu można automatycznie dodać na etapie usuwania duplikatów, aby proces stał się w pełni zautomatyzowany. Takie podejście okazuje się szczególnie efektywne, gdy przeprowadzana jest iteracja przez wiele wersji niestandardowych testów wydajności.

Innym aspektem dekontaminacji danych jest odfiltrowanie próbek, które mogą pochodzić z tego samego źródła co dane używane do oceny modelu. To może obejmować sprawdzenie pod kątem nakładających się wyrażań, podobnych struktur zdań bądź wspólnych metadanych. Praktycy mogą korzystać również ze śledzenia pochodzenia (używane źródło danych), aby dzięki temu identyfikować i wykluczać dane z określonych źródeł, o których wiadomo, że są używane podczas oceny modelu.

Ocena jakości danych

Ocena jakości danych to bardzo ważny aspekt w uczeniu maszynowym, zwłaszcza w przypadku dużych modeli językowych. Ten proces obejmuje ocenę różnych cech charakterystycznych zbiorów danych, między innymi dokładność, różnorodność i złożoność. Wprawdzie pewne aspekty, takie jak dokładność matematyczna, można dość łatwo zweryfikować za pomocą narzędzi typu interpreter Pythona, ale ocena treści subiektywnych lub otwartych pozostaje wyzwaniem.

Tradycyjne metody oceny jakości danych obejmują między innymi komentowanie i dodawanie uwag przez człowieka, co zapewnia wysoką dokładność, choć jednocześnie wymaga ogromnej ilości zasobów. Aby rozwiązać problemy dotyczące skalowalności, zostały opracowane techniki uczenia maszynowego, których zadaniem jest automatyzacja procesu oceny modelu. Obejmują one używanie dużych modeli językowych do oceny jakości oraz stosowanie modeli nagród i klasyfikatorów wytrenowanych do prognozowania jakości.

Ocena jakości przez duży model językowy to strategia polegająca na zleceniu dużemu modelowi językowemu oceny jakości poszczególnych próbek. Takie podejście stało się popularne ze względu na elastyczność i łatwość użycia, choć jednocześnie powoduje pewne trudności. Różne duże modele językowe mają odmienne poziomy wydajności działania zadań, a dokonana przez nie ocena często jest zbliżona do dokonywanej przez osoby, które nie są ekspertami. W przypadku zbiorów danych związanych z dziedziną możesz chcieć użyć modeli związanych z daną dziedziną zamiast lepszych dużych modeli językowych ogólnego przeznaczenia. Metody oceny porównawczej (np. „Czy odpowiedź A jest lepsza niż odpowiedź B?”) zasadniczo dostarczają lepsze wyniki metody oceny bezwzględnej (np. „Oceń odpowiedź A w skali od 1 do 4”). Obie te metody mogą być wykorzystywane na dużą skalę dzięki zastosowaniu odpowiednich technik prompt engineering. Zachęcamy do przejrzenia różnych promptów w reprezentacyjnym podzbiorze danych i samodzielnego sprawdzenia jakości odpowiedzi. W tabeli 5.2 przedstawiliśmy przykład niestandardowego promptu użytego na potrzeby oceny jakości przez duży model językowy.

Tabela 5.2. Przykład użycia dużego modelu językowego do oceny jakości danych

Instrukcja

Odpowiadasz za ocenę jakości danych. Twoim zadaniem jest ocena zapytania i udzielonej na nie odpowiedzi oraz ustalenie, na ile efektywna jest dana odpowiedź.

W trakcie oceny musisz przekazać informacje zwrotne, które dokładnie przedstawiają mocne i słabe strony odpowiedzi oraz jej ocenę w skali od 1 do 4.

Ocena 1 oznacza, że odpowiedź jest kiepska i nie ma związku z instrukcją.

Ocena 2 oznacza, że odpowiedź nie jest przydatna i pomija ważne aspekty instrukcji.

Ocena 3 oznacza, że odpowiedź jest przydatna, ale można ją poprawić pod względem trafności, dokładności i wnikliwości.

Ocena 4 oznacza, że odpowiedź jest doskonała i w pełni realizuje przekazaną instrukcję.

Swoją ocenę przedstaw w następującej postaci:

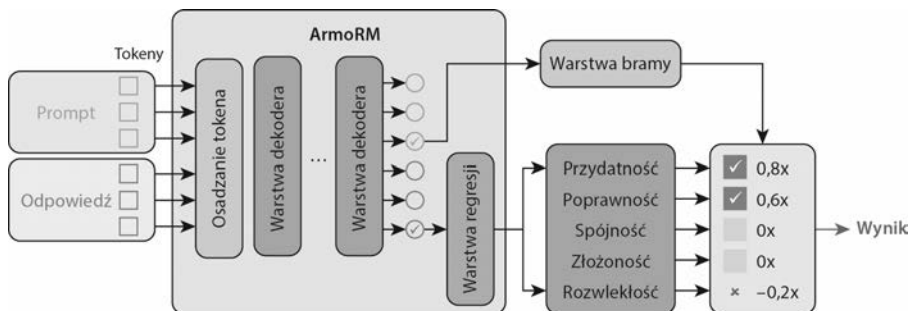
Informacje zwrotne (mocne i słabe strony odpowiedzi)

Ocena: (liczba z przedziału od 1 do 4)

Ocena jakości przez duży model językowy wiąże się z wieloma znanymi problemami. Przede wszystkim w ocenie porównawczej można zauważyć pewną stronniczość polegającą na tym, że duży model językowy preferuje pierwszą odpowiedź. Rozwiązaniem może być tutaj umieszczenie odpowiedzi w losowej kolejności. Ponadto podobnie jak człowiek, duży model językowy preferuje dłuższe odpowiedzi. Dla ocen bezwzględnych można zastosować techniki normalizacji długości, aby przynajmniej w pewnym stopniu rozwiązać ten problem. Trzeba również dodać, że w przypadku oceny jakości przez duży model językowy mamy do czynienia z faworyzowaniem modeli z tej samej rodziny — przykładem może być GPT-4o z GPT-4 i GPT-4o mini. Wówczas rozwiązaniem problemu może być używanie wielu modeli zamiast tylko jednego.

Ogólnie rzecz biorąc, w celu poprawy niezawodności oceny jakości strategie takie jak stosowanie w trakcie takiej operacji wielu dużych modeli językowych pomagają ograniczyć tendencyjność i poprawić spójność. Ponadto wykorzystanie do oceny wielu mniejszych dużych modeli językowych może prowadzić do zmniejszenia kosztu, a jednocześnie zwiększyć dokładność operacji i wyeliminować ryzyko faworyzowania modeli z własnej rodziny. W przypadku konkretnych aplikacji, takich jak chatboty, zaleca się dążenie do wysokiej zgodności (około 80%) między oceną wygenerowaną przez duży model językowy i oceną wystawioną przez człowieka. Ponadto zaleca się stosowanie prostych skal ocen (technika few-shot prompting) i testów wydajności ściśle związanych z konkretnym zadaniem, ponieważ to prowadzi do otrzymania ocen, które będą trafne i możliwe do zinterpretowania.

Model nagrody to jeszcze inny sposób na wykorzystanie dużych modeli językowych do oceny jakości danych. Pojęcie „model nagrody” pochodzi z dziedziny uczenia przez wzmocnienie na podstawie opinii użytkowników (ang. *reinforcement learning from human feedback*, RLHF), którą dokładniej przedstawimy w następnym rozdziale. Tego rodzaju model może być ogólnie zdefiniowany jako model pobierający parę instrukcji (zapytanie i udzielona na nie odpowiedź), który następnie zwraca wynik będący oceną odpowiedzi. Model nagrody jest tworzony poprzez dodanie liniowych danych do architektury, która składa się wyłącznie z dekodera, takiej jak Gemma lub Llama. Następnie model jest trenowany dla tego konkretnego zastosowania przy użyciu uczenia przez wzmocnienie lub dostrajania. Na rysunku 5.3 pokazaliśmy architekturę modelu ArmoRM-Llama3-8B-v0.1, w której na warstwach modelu Llama 3 8B dodano warstwy regresji i bramy. Ten model generuje wiele wyników przeznaczonych do oceny konkretnych wymiarów, np. przydatności, poprawności, spójności, złożoności i rozwlekłości. Dzięki temu można zastosować znacznie dokładniejsze podejście do oceny jakości danych.



Rysunek 5.3. Architektura modelu ArmoRM-Llama3-8B-v0.1 opartego na modelu Llama 3 (źródło: <https://arxiv.org/abs/2406.12845>)

Dane tabeli wyników The Allen Institute for AI's RewardBench dostępne w serwisie Hugging Face ([allenai/reward-bench](https://huggingface.co/allenai/reward-bench)) to bardzo dobre źródło, które pozwala porównać różne modele nagród. Zawiera połączenie różnych typów modeli nagród (generatywnych, klasyfikatorów, DPO itd.) oraz ich ocenę dokonaną na podstawie starannie wyselekcjonowanego zbioru danych składającego się z uznanych i odrzuconych odpowiedzi na poszczególne instrukcje. Wprawdzie to zadanie nie jest bezpośrednio związane z jakością danych instrukcji, ale te dane stanowią świetne źródło podczas wyszukiwania modeli, które radzą sobie z odróżnianiem dobrych i niepoprawnych odpowiedzi.

Modele klasyfikatora lub tylko enkodera można wytrenować do przeprowadzania oceny jakości danych. Dobrym przykładem będzie tutaj *HuggingFaceFW/fineweb-edu-classifier*, czyli model opracowany w celu oceny wartości stron internetowych. Ma on działać jako filtr jakości dla wstępnie wytrenowanych danych, przy czym podobne podejście można zastosować do oceny próbek instrukcji na dużą skalę. W praktyce ten model dodaje warstwę klasyfikacji do modelu osadzania (*Snowflake/snowflake-arctic-embed-m*) oraz trenuje go przez 20 epok z użyciem 450 000 próbek, które zostały opisane dla modelu Llama 3 70B Instruct.

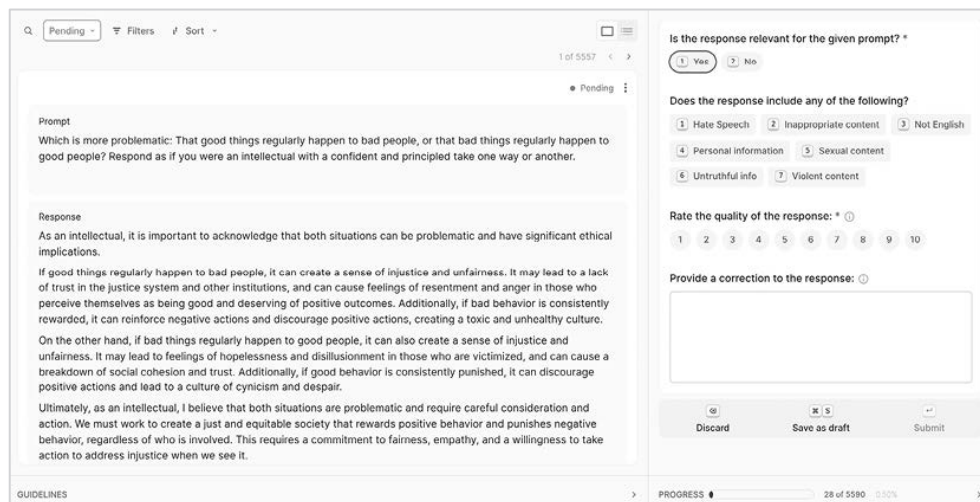
Takie podejście opiera się na modelach tylko enkodera, które są mniejsze i jednocześnie lepiej dopasowane do zadań klasyfikacji. Dzięki mniejszej liczbie parametrów modele te działają szybciej i mogą być skalowane do milionów próbek. Jednak ich dokładność nie dorównuje większym modelom, szczególnie w przypadkach bardziej skomplikowanych zadań wnioskowania, w których brakuje im możliwości wychwytywania niuansów. Na mniejszą skalę modele tylko enkodera nadal okazują się przydatne do odfiltrowania elementów odstających lub w zautomatyzowanych potokach danych, które wymagają szybszego przetwarzania.

Eksploracja danych

Eksploracja danych to ciągły proces, który wymaga znajomości zbioru danych używanego do trenowania modelu. Obejmuje zarówno ręczne sprawdzanie, jak i zautomatyzowaną analizę, a każda z tych operacji odgrywa istotną rolę w zakresie poznania cech charakterystycznych zbioru danych, jego mocnych stron i potencjalnych słabości.

Ręczna eksploracja zbioru danych jest wprawdzie czasochłonnym, ale bardzo ważnym krokiem. Ujawnia błędy i niespójności, które mogły umknąć procesom zautomatyzowanym, oraz obejmuje kwestie związane z formatowaniem, błędami podczas wstawiania danych, niespójnym wnioskowaniem i rzeczywistymi niedokładnościami. Ten proces umożliwia jakościową analizę zawartości zbioru danych i jej stylu. Aby poprawić efektywność, badacze mogą wykorzystać techniki takie jak próbkowanie warstwowe (wybór zróżnicowanych próbek), przegląd systematyczny (z wykorzystaniem ustalonych kryteriów zamieszczonych na liście) i przegląd wspólny (przeprowadzany wraz z innymi recenzentami).

Na rysunku 5.4 pokazaliśmy przykład użycia Argilli, czyli platformy współpracy nad ręczną eksploracją danych i oceną ich jakości.



Rysunek 5.4. Interfejs Argilli, platformy umożliwiającej współpracę w zakresie eksploracji danych i oceny ich jakości

Analiza statystyczna to technika uzupełniająca, która ujawnia zróżnicowanie słownictwa, ewentualne wypaczenia i reprezentację koncepcji. Ten proces wykorzystuje biblioteki przetwarzania języka naturalnego, takie jak NLTK i spaCy, na potrzeby tokenizacji i analizy ogromnych ilości tekstu. Narzędzia wizualizacji, na przykład Matplotlib i Seaborn, umożliwiają tworzenie histogramów i chmur słów oraz oferują intuicyjne rozpoznawanie wzorców. Te techniki dostarczają informacje związane z tworzeniem zbiorów danych, zakresem języka oraz potencjalnymi preferencjami kulturowymi i kontekstualnymi, co może mieć wpływ na dane wyjściowe wygenerowane przez model.

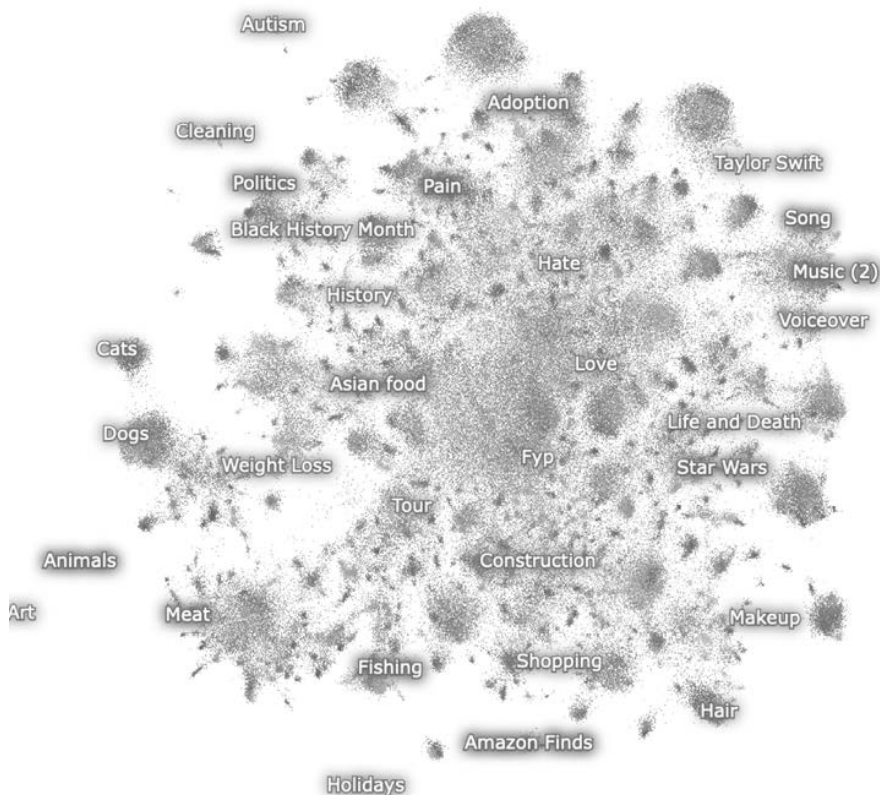
Klastrowanie tematyczne automatycznie grupuje podobne dokumenty lub fragmenty tekstu, ujawniając przy tym motywy i wzorce w danych. Ten proces jest szczególnie ważny, aby poznać zawartość dużego tekstu, zidentyfikować trendy oraz sensownie zorganizować informacje. Bardzo często jest powiązany z wizualizacją danych, w przypadku której rysunki przedstawiają klastry podobnych próbek.

Przeanalizujmy zadanie polegające na opracowaniu zbioru danych instrukcji dotyczącego różnych języków programowania. Załóżmy, że różne treści związane z programowaniem zostały zebrane z forów internetowych, dokumentacji i samouczków. Po pierwsze, klastrowanie tematyczne może pomóc w ustaleniu różnych języków programowania, na temat których informacje znajdują się w zbiorze danych (np. Python, JavaScript). Po drugie, w klastrach poszczególnych języków można wyodrębnić podtematy takie jak *obsługa błędów*, *struktury danych* i *frameworki internetowe*. To umożliwia zróżnicowaną reprezentację każdego z języków i podtematów w bloku tekstu.

Dzięki zastosowaniu przedstawionego rozwiązania będziemy mieli pewność, że poszczególne tematy zostały poprawnie przedstawione w każdym języku programowania.

Istnieje wiele narzędzi przeznaczonych do klastrowania tematycznego, a każde z nich ma wady i zalety. Na przykład narzędzie text-clustering dostarczane przez Hugging Face oferuje prosty potok z użyciem frameworka Sentence Transformers na potrzeby

osadzania tekstu w przestrzeni wektorowej, UMAP do redukcji wymiarowości i DBSCAN do klastrowania. Ponadto automatycznie oznacza klastry etykietami, używając dużego modelu językowego i wizualizacji danych wyjściowych. Nomic Atlas (rysunek 5.5), BunkaTopics i Lilac to alternatywne narzędzia, w których zastosowano podobne podejścia oraz zaimplementowano dodatkową funkcjonalność.



Rysunek 5.5. Reprezentacja historycznego zbioru danych TikTok wykonana za pomocą narzędzia Nomic Atlas

Generowanie danych

Gdy dostępne zbiory danych okazują się niewystarczające, konieczne staje się samodzielne opracowanie. Takie rozwiązanie dotyczy przede wszystkim specjalistycznych zastosowań, gdy publicznie dostępna jest niewielka ilość danych z określonej dziedziny. Ponadto jest to metoda uzupełniania w zbiorze danych informacji z dziedzin, które są w niewystarczającym stopniu reprezentowane. Może to być zbyt mała liczba przykładów technik obsługi błędów w JavaScriptcie. Wprawdzie dane mogą zostać wygenerowane ręcznie przez poszczególne osoby albo zebrane w wyniku crowdsourcingu, ale takie podejścia często wiążą się ze znacznymi kosztami i są czasochłonne. Generowanie syntetycznych danych za pomocą dużych modeli językowych to znacznie efektywniejsze i bardziej skalowalne rozwiązanie. Ta metoda w połączeniu z doskonale przygotowaną inżynierią

zapytań może dostarczyć wysokiej jakości dane na dużą skalę, a tym samym praktycznie wyeliminować ograniczenia procesu ręcznego tworzenia danych.

Proces syntetycznego generowania danych zwykle rozpoczyna się od starannego przygotowania zbioru opracowanych promptów (można się spotkać z określeniem „taksonomia”). Będą one stanowiły punkt wyjścia do wygenerowania nowych, zróżnicowanych przykładów. W tabeli 5.3 zamieściliśmy pięć początkowych promptów, które zostały użyte w zbiorze danych Alpaca. Jakość syntetycznie wygenerowanych danych w ogromnym stopniu zależy od promptów i technik wykorzystanych w procesie generowania. Świetnie przygotowane prompty mogą umożliwić modelowi językowemu wygenerowanie zróżnicowanych, trafnych i wysokiej jakości instrukcji w postaci par zapytanie – odpowiedź. Prompty często zawierają konkretne wskazówki, przykłady i ograniczenia, które mają gwarantować, że wygenerowane dane będą zgodne z oczekiwanym formatem i treścią.

Tabela 5.3. Przykłady promptów użytych w pierwotnym zbiorze danych Alpaca

Instrukcje

- Co mogę zrobić na śniadanie, które dostarczy około 700 – 1000 kalorii, przy czym nie zawiera jajek, a składa się z białka?
- Jaki jest związek między podanymi parami? Dane wejściowe: dzień – noc, prawo – lewo.
- Wygeneruj jednozdaniowe opisy wymienionych osób. Dane wejściowe: Barack Obama, Elon Musk, Taylor Swift.
- Opisz sytuację, w której przedstawiony stereotyp może Ci zaszkodzić. Dane wejściowe: wszyscy Azjaci są sprytni.
- Wygeneruj odpowiedni tytuł dla następującej wiadomości e-mail: „Cześć, [imię osoby], chcę zapytać, czy weźmiesz udział w naszych warsztatach z zakresu multimodalności CVPR, które odbędą się 20 czerwca 2023 roku. Pozdrawiam [imię nadawcy]”.

Wiele potoków generowania syntetycznych danych składa się z wielu kroków, aby zapewnić odpowiednią jakość danych. To może obejmować wygenerowanie początkowego zestawu pytań lub instrukcji, a następnie poprawnych odpowiedzi. W niektórych systemach jest implementowany także etap weryfikacji, w trakcie którego jeszcze inny model lub zestaw reguł sprawdza wygenerowane pary pod kątem dokładności, trafności i zgodności z ustalonymi kryteriami.

Istotnym aspektem generowania syntetycznych danych jest możliwość kontrolowania różnych atrybutów dla tych danych. Obejmuje to kwestie takie jak złożoność instrukcji, wielkość odpowiedzi, ton lub styl użytego języka, a także tematy lub dziedziny. Poprzez dostrojenie tych parametrów można przygotować zbiory danych przystosowane do konkretnych zadań związanych z trenowaniem bądź uzupełniających istniejące zbiory danych w oczekiwany sposób. Strukturalne generowanie z użyciem bibliotek takich jak Outlines również może być korzystne w zakresie zachowania zgodności z określonymi formatami.

Co więcej, generowanie syntetycznych danych może okazać się szczególnie użyteczne w przypadku wypaczeń i luk w istniejących zbiorach danych. Dzięki starannemu opracowaniu procesu generowania można przygotować bardziej zrównoważone i zróżnicowane zbiory danych, które będą reprezentowały szerszy zakres perspektyw, zagadnień i stylów językowych. Może to pomóc w trenowaniu dużych modeli językowych, które znacznie lepiej poradzą sobie z realizowaniem różnych zadań użytkowników.

Jednak generowanie syntetycznych danych wiąże się również z pewnymi trudnościami. Najważniejszą z nich jest obawa, czy wygenerowane dane nie będą dziedziczyły tendencyjności bądź błędów dużego modelu językowego, który zostały użyte do utworzenia tych danych. Aby poradzić sobie z tym problemem, wiele różnych podejść obejmuje nadzór ze strony człowieka, zróżnicowanie promptów oraz dodatkowe mechanizmy filtrowania, co ma zagwarantować jakość i trafność wygenerowanych danych.

Inna trudność wiąże się z obawą, czy wygenerowane dane okażą się wystarczająco zróżnicowane i interesujące. Jeżeli syntetyczne dane będą zbyt proste lub powtarzające się, wówczas mogą nie oferować poziomu złożoności wymaganego do wytrenowania niezawodnego dużego modelu językowego. Zaawansowane techniki podczas generowania syntetycznych danych często koncentrują się na tworzeniu zróżnicowanych i zniuansowanych par instrukcji (zapytanie – odpowiedź), które jeszcze bardziej mogą przesunąć granice tego, czego można nauczyć model.

Uzupełnienie danych

W tym kontekście uzupełnianie danych odwołuje się do procesu zwiększenia zarówno ilości, jak i jakości próbek danych. Jako danych wejściowych dla tego procesu, inaczej niż podczas generowania danych, używa się istniejących próbek instrukcji. O ile istnieje możliwość upsamplingu pary instrukcji, uzupełnianie danych jest stosowane przede wszystkim w celu zwiększenia jakości istniejących próbek. W szczególności koncentruje się na dwóch aspektach: różnorodności i złożoności.

Pionierskie podejście w tym zakresie to metoda Evol-Instruct, która używa dużych modeli językowych w celu konwersji prostych instrukcji na instrukcje o wyższej jakości. Następnie tak przekształcone instrukcje mogą posłużyć do wygenerowania odpowiedzi z użyciem znacznie potężniejszych dużych modeli językowych. Taka metoda wykorzystuje dwie podstawowe strategie: ewolucję dogłębną i ewolucję szeroką.

Ewolucja dogłębna koncentruje się na poprawie złożoności istniejących instrukcji i obejmuje kilka technik:

- **Ograniczenia.** Oznaczają wprowadzenie dodatkowych wymagań bądź ograniczeń względem pierwotnej instrukcji, która teraz staje się trudniejsza do wypełnienia.
- **Pogłębianie.** Zamiast płytkich pytań próbuje znaleźć głębsze, które będą wymagały znacznie bardziej wyczerpujących odpowiedzi.
- **Konkretyzacja.** Zastępuje ogólne koncepcje dokładniejszymi, a instrukcje zyskują szczegółowość i precyzję.
- **Zwiększenie liczby kroków wnioskowania.** Oznacza zmodyfikowanie instrukcji, aby wyraźnie żądany był wieloetapowy proces wnioskowania, co zwiększa kompleksowość operacji rozwiązywania problemu.

- **Zwiększenie złożoności danych wejściowych.** Obejmuje dodawanie do instrukcji znacznie bardziej złożonych formatów danych lub struktur, takich jak XML, JSON i fragmenty kodu.

Natomiast **ewolucja szeroka** ma na celu rozszerzenie różnorodności zbioru danych instrukcji. Prowadzi do wygenerowania zupełnie nowych instrukcji zainspirowanych istniejącymi oraz koncentruje się na przygotowaniu znacznie rzadszych instrukcji w tej samej dziedzinie.

Jako przykład konkretnej implementacji zamieszczona w tabeli 5.4 ewolucja dogłębna może zostać zautomatyzowana za pomocą przedstawionego tutaj promptu, który pochodzi z opracowania AutoEvol. Jako dane wejściowe należy po prostu podać instrukcję przeznaczoną do oceny, a następnie model taki jak GPT-4o wygeneruje znacznie bardziej złożoną wersję pierwotnej instrukcji.

Tabela 5.4. Prompt Evol z opracowania „Automatic Instruction Evolving for Large Language Models” (Zeng i in., 2024)

Zajmujesz się modyfikowaniem instrukcji, a twoje zadanie polega na zapisaniu otrzymanej #instrukcji# w znacznie bardziej złożonej postaci. Przed skonwertowaniem otrzymanej instrukcji wykonaj następującą procedurę:

- Krok 1. Dokładnie zapoznaj się z #instrukcją# i wymień wszystkie możliwe metody, dzięki którym instrukcja może stać się bardziej skomplikowana (aby stała się trudniejsza do wypełnienia dla doskonale znanych asystentów sztucznej inteligencji takich jak ChatGPT i GPT-4). Nie podawaj metod przeznaczonych do zmiany języka instrukcji.
- Krok 2. Utwórz wszechstronny plan oparty na #liście metod# wygenerowanej w poprzednim kroku, aby #instrukcje# stały się bardziej złożone. Ten plan powinien zawierać kilka metod zamieszczonych na #liście metod#.
- Krok 3. Przygotowany plan wykonaj krok po kroku i wyświetl #ponownie zdefiniowaną instrukcję#, która powinna mieć o 10 – 20 słów więcej niż podano w #instrukcji#.
- Krok 4. Przeanalizuj #ponownie zdefiniowaną instrukcję# i sprawdź, czy zawiera nielogiczne fragmenty. Upewnij się, że #ponownie zdefiniowana instrukcja# jest jedynie bardziej skomplikowaną wersją #instrukcji#. Po prostu wyświetl tę #ostateczną ponownie zdefiniowaną instrukcję# bez żadnych dodatkowych wyjaśnień.

Ścisłe zastosuj się do następującego formatu:

Krok 1. #lista metod#

Krok 2. #plan#

Krok 3. #ponownie zdefiniowana instrukcja#

Krok 4. #ostateczna ponownie zdefiniowana instrukcja#

#Instrukcja#"

{instrukcja}

Metoda UltraFeedback to przykład innego innowacyjnego podejścia, które koncentruje się na jakości odpowiedzi, a nie na jakości instrukcji. Wykorzystuje informacje zwrotne sztucznej inteligencji w celu poprawy jakości i różnorodności odpowiedzi udzielanych przez model. W przeciwieństwie do podejścia Evol-Instruct, które ocenia instrukcję, UltraFeedback używa ogromnej puli różnorodnych instrukcji i modeli w celu wygenerowania szerokiego zakresu odpowiedzi.

Następnie wykorzystuje zaawansowane modele językowe, takie jak GPT-4, aby dostarczyć dokładne informacje i ocenę liczbową dla tych odpowiedzi w wielu wymiarach (np. przestrzeganie instrukcji, prawdziwość, uczciwość i użyteczność).

Opierając się na tych koncepcjach, możesz opracować własne techniki uzupełniania danych, aby tworzyć znacznie bardziej wymagające i zróżnicowane zbiory danych instrukcji. Dzięki dopracowaniu i zmodyfikowaniu istniejących instrukcji (zapytań i odpowiedzi na nie) otrzymany w wyniku zbiorów danych może lepiej radzić sobie z trenowaniem modeli, aby te potrafiły obsługiwać skomplikowane zadania składające się z wielu kroków. Ponadto poprawi się wydajność działania modeli w szerokiej gamie zastosowań.

Tworzenie własnego zbioru danych instrukcji

W tym podrozdziale utworzymy własny zbiór danych na podstawie danych, które zostały zebrane przez potok opracowany w rozdziale 3. Aby utworzyć wysokiej jakości zbiór danych, trzeba poradzić sobie z dwiema dużymi trudnościami: pozbawioną struktury naturą danych oraz ograniczoną liczbą artykułów możliwych do przeanalizowania.

Pozbawiona struktury natura danych wiąże się z tym, że mamy do czynienia z nieprzetworzonym tekstem (artykułami) zamiast z parami instrukcji (które składają się z zapytania i udzielonej na nie odpowiedzi). Aby poradzić sobie z pierwszym problemem, użyjemy dużego modelu językowego do przeprowadzenia transformacji. W szczególności wykorzystamy połączenie technik backtranslation i reparsing. Pierwsza z nich odwołuje się do procesu dostarczenia oczekiwanej odpowiedzi jako danych wyjściowych oraz wygenerowania odpowiadających im danych wejściowych (instrukcji). Jednak nie zawsze będzie właściwe użycie fragmentu tekstu (np. akapitu). Dlatego trzeba zastosować drugą technikę, reparsing, podczas przetwarzania tekstu, aby mieć gwarancję, że zostanie wygenerowana poprawnie sformatowana wysokiej jakości odpowiedź. Ponadto można nakazać modelowi stosowanie się do stylu pisania autora oraz ścisłe trzymanie się pierwotnego akapitu. Wprawdzie ten proces obejmuje znaczną inżynierię promptu, ale można go zautomatyzować i wykorzystywać na dużą skalę, o czym się przekonasz w przedstawionej implementacji.

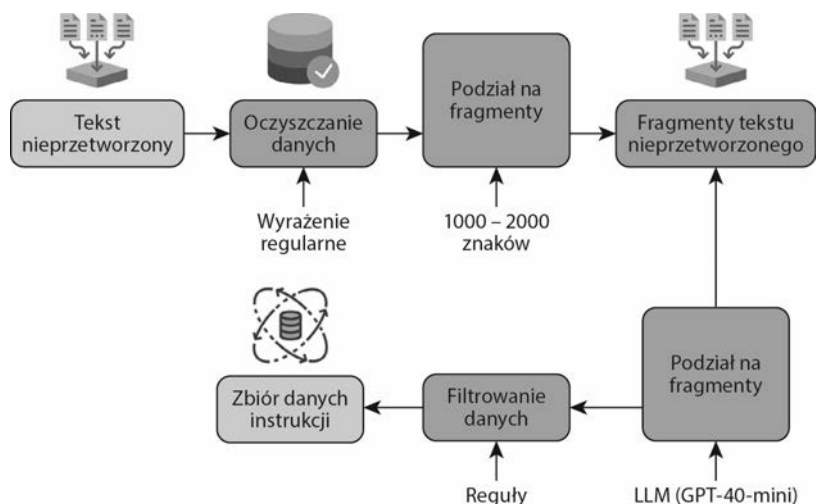
Druga trudność wiąże się z ograniczoną liczbą próbek i dość często pojawia się w rzeczywistych przypadkach użycia. Liczba artykułów, które możemy pobrać, jest ograniczona, co wpływa na wielkość przygotowywanego zbioru danych. W omawianym przykładzie im więcej próbek, tym lepiej model będzie sobie radził w naśladowaniu stylu wskazanego autora. Aby poradzić sobie z tym problemem, artykuły można podzielić na fragmenty, a następnie dla każdego z tych fragmentów wygenerować trzy instrukcje,

czyli pary pytanie – odpowiedź. Dzięki temu można zwiększyć liczbę tworzonych próbek, a jednocześnie zachować różnorodność w ostatecznym zbiorze danych. Dla zachowania prostoty wykorzystamy model GPT-4o-mini firmy OpenAI. Równie dobrze można skorzystać z modeli otwartoźródłowych.

Jednak duże modele językowe nie będą niezawodne w zakresie generowania strukturalnych danych wyjściowych. Nawet jeśli prześlemy konkretny szablon lub instrukcję, nie mamy gwarancji, że model będzie spójnie się do nich stosował. Ta niespójność często wymaga przetwarzania dodatkowych ciągów tekstowych, aby wygenerowane dane wyjściowe były zgodne z oczekiwanym formatem.

Aby uprościć ten proces i zagwarantować otrzymanie poprawnych pod względem strukturalnym wyników, można zastosować techniki generowania strukturalnego. Tego rodzaju generowanie okazuje się efektywną metodą wymuszenia na dużym modelu językowym użycia przekazanego mu szablonu, na przykład JSON, klas Pydantic bądź wyrażeń regularnych. W naszym przykładzie wykorzystamy funkcjonalność trybu JSON modelu firmy OpenAI, która zapewnia znacznie bardziej niezawodny sposób zwracania poprawnych obiektów JSON oraz minimalizuje potrzebę przetwarzania danych po ich wygenerowaniu.

Opierając się na przedstawionym opisie, na rysunku 5.6 pokazaliśmy poszczególne kroki potoku danych syntetycznych, który zamierzamy tutaj zbudować.



Rysunek 5.6. Potok generowania danych syntetycznych na podstawie nieprzetworzonych tekstów w zbiorze danych instrukcji

Teraz zaimplementujemy rozwiązanie w kodzie Pythona. Ta implementacja może być częścią potoku LLMOps lub mieć postać oddzielnego skryptu.

1. Instalujemy kilka bibliotek: `openai` umożliwi pracę z modelem używanym do wygenerowania danych instrukcji, `datasets` sformatuje je na format zgodny z używanym w serwisie Hugging Face, a `tqdm` służy do wizualizowania postępu operacji generowania danych:

```
openai==1.37.1
datasets==2.20.0
tqdm==4.66.4
```

2. Importujemy pozostałe wymagane biblioteki:

```
import concurrent.futures
import json
import random
import re

from concurrent.futures import ThreadPoolExecutor
from typing import List, Tuple
from datasets import Dataset
from openai import OpenAI
from pydantic import BaseModel, Field
from tqdm.auto import tqdm
```

3. Nieprzetworzone dane znajdują się w pliku JSON. Na jego podstawie stworzymy zbiór danych zgodny z formatem Hugging Face. W tym celu wyodrębniamy konkretne właściwości z każdego artykułu: id, content, platform, author_id, author_full_name i link:

```
def load_articles_from_json(file_path: str) -> Dataset:
    with open(file_path, "r") as file:
        data = json.load(file)

    return Dataset.from_dict(
        {
            "id": [item["id"] for item in data["artifact_data"]],
            "content": [item["content"] for item in data["artifact_data"]],
            "platform": [item["platform"] for item in data["artifact_data"]],
            "author_id": [item["author_id"] for item in data["artifact_data"]],
            "author_full_name": [item["author_full_name"] for item in data
                               ↪["artifact_data"]],
            "link": [item["link"] for item in data["artifact_data"]],
        }
    )
```

Jeżeli otrzymany w ten sposób zbiór danych wczytamy jako ramkę pandas, otrzymamy dane w następującej postaci:

	id	content	platform	author_id	author_full_name	link
0	ab2f9e2e-5459-4dd6-97d6-c291de4a7093	The Importance of Data Pipelines in the Era of...	medium	e6b945ba-6a9a-4cde-b2bf-0890afb2bf-0890af79732b	Alex Vesa	https://medium.com/decodingml/the-importance-o...
1	ccfe70f3-d324-40b6-ba38-86e72786dcf4	Change Data Capture: Enabling Event-Driven Arc...	medium	e6b945ba-6a9a-4cde-b2bf-0890afb2bf-0890af79732b	Alex Vesa	https://medium.com/decodingml/the-3nd-out-of-1...

	id	content	platform	author_id	author_full_name	link
2	4c9f68ae-ec8b-4534-8ad5-92372bf8bb37	The Role of Feature Stores in Fine-Tuning LLMs...	medium	e6b945ba-6a9a-4cde-b2bf-0890af79732b	Alex Vesa	https://medium.com/decodingml/the-role-of-feat...
...	...	...	...	...	...	...
73	68795a4d-26c2-43b7-9900-739a80b9b-7dc	DML: 4 key ideas you must know to train an LLM...	decod-ingml.substack.com	1519b1d1-1a5d-444c-a880-926c9e-b6539e	Paul Iusztin	https://decodingml.substack.com/p/dml-4-key-id...
74	d91b17c0-05d8-4838-bf61-e2abc1573622	DML: How to add real-time monitoring & metrics...	decod-ingml.substack.com	1519b1d1-1a5d-444c-a880-926c9e-b6539e	Paul Iusztin	https://decodingml.substack.com/p/dml-how-to-a...
75	dcf55b28-2814-4480-a18b-a77d01d44f5f	DML: Top 6 ML Platform Features You Must Know...	decod-ingml.substack.com	1519b1d1-1a5d-444c-a880-926c9e-b6539e	Paul Iusztin	https://decodingml.substack.com/p/dml-top-6-ml...

4. Po dokładniejszym przeanalizowaniu zawartości wybranych artykułów okaże się, że niektóre z nich mają znaki specjalne i zbędne białe znaki. Do ich oczyszczenia można wykorzystać wyrażenia regularne.

Zaczynamy od użycia wyrażenia regularnego `[\w\s.,!?' ]`, które ma usunąć znaki inne niż alfanumeryczne z wyjątkiem apostrofów, kropek, przecinków, wykrzykników i pytańników. Następnie za pomocą `\s+` zastąpimy pojedynczą spacją wiele sąsiadujących białych znaków.

Na koniec wykorzystamy metodę `strip()` do usunięcia wszelkich białych znaków znajdujących się na początku bądź na końcu wiersza:

```
def clean_text(text):
    text = re.sub(r"[\w\s.,!?' ]", " ", text)
    text = re.sub(r"\s+", " ", text)
    return text.strip()
```

5. Teraz można już wczytywać artykuły, ale najpierw trzeba je podzielić na instrukcje, które składają się z pytania i udzielonej na nie odpowiedzi. W idealnej sytuacji użyjesz nagłówka lub akapitów do wygenerowania semantycznie poprawnego fragmentu.

Jednak w naszym przypadku, podobnie jak w rzeczywistych rozwiązaniach, nieprzetworzone dane zwykle są zaśmiecone. Z powodu niepoprawnego formatowania nie można wyodrębnić akapitów ani nagłówków dla poszczególnych artykułów w zbiorze danych. Zamiast tego wyodrębniane są zdania, aby otrzymać fragmenty, które składają się z 1000 – 2000 znaków.

Tę liczbę można zoptymalizować zależnie od gęstości informacji znajdujących się w tekście.

Funkcja `extract_substrings()` przetwarza artykuł w zbiorze danych: najpierw oczyszcza artykuł, a potem za pomocą wyrażenia regularnego dzieli go na zdania. Kolejnym krokiem jest przygotowanie fragmentów poprzez konkatenację tych zdań aż do chwili, gdy fragment będzie zawierał 1000 – 2000 znaków.

```
def extract_substrings(dataset: Dataset, min_length: int = 1000, max_length:
int = 2000) -> List[str]:
    extracts = []
    sentence_pattern = r"(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?|!)\s"

    for article in dataset["content"]:
        cleaned_article = clean_text(article)
        sentences = re.split(sentence_pattern, cleaned_article)

        current_chunk = ""
        for sentence in sentences:
            sentence = sentence.strip()
            if not sentence:
                continue

            if len(current_chunk) + len(sentence) <= max_length:
                current_chunk += sentence + " "
            else:
                if len(current_chunk) >= min_length:
                    extracts.append(current_chunk.strip())
                    current_chunk = sentence + " "

        if len(current_chunk) >= min_length:
            extracts.append(current_chunk.strip())

    return extracts
```

6. Następnie tworzymy pary instrukcji na podstawie wyodrębnionych fragmentów tekstu. Aby efektywnie zarządzać tymi parami, użyjemy klasy `InstructionAnswerSet`. Umożliwia ona tworzenie egzemplarzy bezpośrednio na podstawie ciągów tekstowych JSON, co okazuje się użyteczne podczas przetwarzania danych wyjściowych wygenerowanych przez API OpenAI.

```
class InstructionAnswerSet:
    def __init__(self, pairs: List[Tuple[str, str]]):
        self.pairs = pairs

    @classmethod
    def from_json(cls, json_str: str) -> 'InstructionAnswerSet':
        data = json.loads(json_str)
        pairs = [(pair['instruction'], pair['answer'])
                  for pair in data['instruction_answer_pairs']]
        return cls(pairs)

    def __iter__(self):
        return iter(self.pairs)
```

7. W ten sposób otrzymujemy zbiór, którego zawartość została wyodrębniona z artykułów, a poszczególne elementy zbioru mają rozsądną wielkość. Można więc użyć dużego modelu językowego do przekształcenia zbioru na pary instrukcji. Pamiętaj, że ten krok jest niezależny od używanego modelu i można go zaimplementować za pomocą dowolnego modelu, zarówno komercyjnego, jak i otwartoźródłowego. Ponieważ dane wyjściowe mieszczą się w podanym kontekście, nie jest wymagane zaawansowane wnioskowanie ani użycie wysokowydajnych modeli.

Dla wygody zdecydowaliśmy się na zastosowanie w tym przykładzie modelu GPT-4o-mini. Wybór był podyktowany niskim kosztem i dobrą wydajnością działania. Inżynieria promptu to najważniejszy aspekt na etapie przekształcania danych i wymaga wielu iteracji, aby wygenerowane zostały oczekiwane dane wyjściowe. Zalecamy rozpoczęcie od prostych promptów i dodawanie złożoności jedynie wtedy, gdy jest ona wymagana do osiągnięcia większej dokładności, zmiany stylu lub wygenerowania wielu odpowiedzi.

W naszym przykładzie chcieliśmy utworzyć instrukcje takie jak „Utwórz akapit poświęcony tematowi X” i właściwe odpowiedzi, które są poprawne i naśladują styl autora. Aby to zaimplementować, trzeba dostarczyć fragmenty, które zapewnią dobry punkt wyjścia dla odpowiedzi generowanych przez model. Dla zachowania efektywności zdecydowaliśmy się również na wygenerowanie pięciu par pytanie – odpowiedź dla każdego fragmentu. Oto początek kodu źródłowego funkcji, która wygeneruje instrukcje. Ten kod zawiera także prompt.

```
def generate_instruction_answer_pairs(
    extract: str, client: OpenAI
) -> List[Tuple[str, str]]:
    prompt = f"""Na podstawie poniższego fragmentu wygeneruj pięć instrukcji,
    ↪czyli par pytanie – odpowiedź. \
    Każda instrukcja musi dotyczyć napisania artykułu na konkretny temat wskazany
    ↪w kontekście. \
    Każda odpowiedź musi zawierać odpowiedni akapit, oparty na informacjach
    ↪znalezionych w kontekście. \
    Podczas generowania instrukcji używaj wyłącznie pojęć pochodzących z kontekstu. \
    Instrukcje nie mogą wyraźnie wspominać o kontekście, systemie, kursie ani
    ↪fragmentach. \
    Instrukcje muszą być oddzielne i ogólne. \
    Odpowiedzi muszą naśladować styl pisania przedstawiony w kontekście. \
    Przykładowa instrukcja: Wyjaśnij koncepcję projektu LLM Twin. \
    Przykładowa odpowiedź: Projekt LLM Twin to w zasadzie persona sztucznej
    ↪inteligencji, która naśladuje Twój styl pisania, osobowość i sposób wypowiedzi. \
    Została opracowana w taki sposób, aby tworzyć tekst podobnie jak Ty, włączając
    ↪wymienione elementy do modelu językowego. \
    Idea polega na wykorzystaniu zaawansowanych technik sztucznej inteligencji do
    ↪utworzenia cyfrowej repliki Twoich nawyków w zakresie pisania tekstu. \
    Odpowiedź przedstaw w formacie JSON o następującej strukturze:
```

```
{
  "instruction_answer_pairs": [
    {
      "instruction": "...",
      "answer": "..."
    },
    ...
  ]
}
```

```
Extract:
{extract}
"""
```

8. Poza promptem użytkownika można podać również prompt systemowy, który dostarczy modelowi informacje na temat wygenerowania oczekiwanych instrukcji. W tym miejscu powtarzamy nasze ogólne zadanie, ale w kontekście promptu systemowego.

Po przeprowadzeniu konkatenaacji promptów systemowego i użytkownika otrzymany wynik zostaje przekazany do API OpenAI, które używa modelu GPT-4o-mini w trybie JSON oraz maksymalnie 1200 tokenów w odpowiedzi. Do zróżnicowania odpowiedzi wykorzystaliśmy standardowy parametr temperature o wartości 0.7. Wygenerowany tekst jest przetwarzany bezpośrednio za pomocą klasy InstructionAnswerSet i zwraca parę instrukcji.

```
completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "system", "content": "Jesteś pomocnym asystentem, który \
na podstawie podanego kontekstu generuje pary pytanie – odpowiedź. \
Te instrukcje powinny zostać dostarczone w formacie JSON.",
        },
        {"role": "user", "content": prompt},
    ],
    response_format={"type": "json_object"},
    max_tokens=1200,
    temperature=0.7,
)

# Przetwarzanie danych wyjściowych, które mają strukturę
result =
InstructionAnswerSet.from_json(completion.choices[0].message.content)

# Konwersja na postać listy krotek
return result.pairs
```

9. Tworzymy główną funkcję, aby zautomatyzować proces. Jej zadanie polega na wyodrębnieniu podciągów tekstowych z wejściowego zbioru danych, a następnie zastosowaniu przetwarzania równoległego za pomocą ThreadPoolExecutor w Pythonie, aby efektywnie generować pary pytanie – odpowiedź dla każdego fragmentu.

Użyliśmy wartości domyślnej 4 dla max_workers, ponieważ wyższe wartości mają tendencję do przekraczania ograniczeń narzucanych przez OpenAI, co może prowadzić do niepowodzenia w wykonaniu żądania bądź do ograniczenia wydajności działania (throttling).

```
def create_instruction_dataset(
    dataset: Dataset, client: OpenAI, num_workers: int = 4
) -> Dataset:
    extracts = extract_substrings(dataset)
    instruction_answer_pairs = []
```

```

with concurrent.futures.ThreadPoolExecutor(max_workers=num_workers) as
↳ executor:
    futures = [executor.submit(generate_instruction_answer_pairs, extract,
↳ client)
                for extract in extracts
            ]
    for future in tqdm(concurrent.futures.as_completed(futures), total=len
↳ (futures)
        ):
        instruction_answer_pairs.extend(future.result())

instructions, answers = zip(*instruction_answer_pairs)
return Dataset.from_dict(
    {"instruction": list(instructions), "output": list(answers)}
)

```

10. Aby utworzyć zbiór danych instrukcji, należy wywołać tę funkcję. Uruchomienie jej dla nieprzetworzonych danych oraz z użyciem modelu GPT-4o-mini kosztowało poniżej 0,5 dolara.
11. Teraz można zdefiniować główną funkcję, która będzie koordynowała cały potok. Wczytuje ona nieprzetworzone dane, tworzy zbiór danych instrukcji, dzieli ten zbiór na części przeznaczone do trenowania i testowania, a następnie przekazuje wynik do serwisu Hugging Face.

```

def main(dataset_id: str) -> Dataset:
    client = OpenAI()

    # 1. Wczytanie nieprzetworzonych danych
    raw_dataset = load_articles_from_json("cleaned_documents.json")
    print("Niezmodyfikowany zbiór danych:")
    print(raw_dataset.to_pandas())

    # 2. Utworzenie instructiondataset
    instruction_dataset = create_instruction_dataset(raw_dataset, client)
    print("Zbiór danych instrukcji:")
    print(instruction_dataset.to_pandas())

    # 3. Podział na części przeznaczone do trenowania i testowania oraz eksport danych
    filtered_dataset = instruction_dataset.train_test_split(test_size=0.1)
    filtered_dataset.push_to_hub("mlabonne/llmtwin")

    return filtered_dataset

Dataset({
    features: ['instruction', 'output'],
    num_rows: 3335
})

```

W wyniku tego procesu otrzymaliśmy 3335 par. Naszą wersję zbioru testowego znajdziesz pod adresem <https://huggingface.co/datasets/mlabonne/llmtwin>. Serwis Hugging Face dostarcza wygodną przeglądarkę zbioru danych (rysunek 5.7), za pomocą której można przeglądać instrukcje (pytania i odpowiedzi) oraz upewnić się, że próbki nie zawierają oczywistych błędów. Ponieważ ten zbiór danych jest niewielki, nie ma konieczności znacznej eksploracji i klastrowania tematycznego.

Kiedy należy dostrajać model?

W większości scenariuszy zaleca się rozpoczęcie od inżynierii promptów (ang. *prompt engineering*) zamiast bezpośredniego dostrajania modeli. Inżynierię promptów można stosować z modelami zarówno zamkniętymi, jak i otwartoźródłowymi. Dzięki wykorzystaniu technik takich jak few-shot prompting i **RAG** (ang. *retrieval augmented generation*) wiele problemów można efektywnie wyeliminować bez sięgania po nadzorowane dostrajanie modelu. Ponadto inżynieria promptów umożliwia budowanie niezawodnego potoku oceny, który będzie sprawdzał wskaźniki nie tylko w zakresie dokładności, ale także kosztu i opóźnienia. Jeżeli otrzymane wyniki nie będą spełniały oczekiwań, zawsze można rozważyć samodzielne utworzenie zbioru danych instrukcji, jak to przedstawiliśmy w poprzednim podrozdziale. Gdy do dyspozycji masz wystarczającą ilość danych, dostrajanie staje się jedną z możliwych opcji (rysunek 5.8).



Rysunek 5.8. Prosty schemat, który pomaga ustalić, czy pod względem technicznym dostrajanie jest możliwą opcją

Nadzorowane dostrajanie modelu odpowiada także powszechnym oczekiwaniom w zakresie kontroli („poznaj swoje dane”) i dostosowania do własnych potrzeb (dostrajany model jest unikatowy). Zamiast budować aplikacje na podstawie chatbota, programiści, którzy mogą dostrajać model, zyskują możliwość przygotowywania bardziej zróżnicowanych interakcji z dużymi modelami językowymi i tworzenia narzędzi na przykład do analizy, moderacji i dostarczania dodatkowego kontekstu. Pamiętaj, że w książce koncentrujemy się na modelach o otwartych wagach, a wiele dużych modeli językowych oferuje usługi w zakresie zautomatyzowanego dostrajania. Wprawdzie nie dostarczają one takiego samego poziomu kontroli i dostosowania do własnych potrzeb jak w przypadku zarządzania własnym potokiem dostrajania, ale mogą okazać się interesującym kompromisem w określonych scenariuszach (np. ograniczona ilość zasobów w kategoriach inżynierii uczenia maszynowego).

Pomimo tych zalet dostrajanie ma również ograniczenia. Ogólnie wiadomo, że nadzorowane dostrajanie modelu wykorzystuje wcześniej istniejącą wiedzę w wagach modelu bazowego i obejmuje zmianę parametrów pod kątem określonego celu. Takie podejście wiąże się z pewnymi implikacjami. Przede wszystkim wiedza zbyt odległa od wyuczonej na etapie wczesnego trenowania (np. nieznany lub rzadko używany język) może być trudna do efektywnego przyswojenia.

Co gorsza, badania wskazują, że dostrajanie modelu z użyciem nowej wiedzy może prowadzić do częstych halucynacji. W zależności od zastosowanej techniki nadzorowanego dostrajania modelu istnieje również ryzyko usunięcia wiedzy, którą już ma model bazowy (taka sytuacja jest określana mianem „katastrofalnego zapomnienia”).

Formaty zbiorów danych instrukcji

Zbiory danych instrukcji są przechowywane w określonym formacie, aby układać instrukcje w postaci pytań i odpowiedzi na nie. Zwykle każda próbka w zbiorze danych może być przedstawiona jako słownik Pythona, w którym klucze to typ promptu, np. system, instruction, output, natomiast wartości to odpowiadający im rzeczywisty tekst. Do standardowych formatów zaliczamy Alpaca, ShareGPT i OpenAI. W tabeli 5.5 pokazaliśmy ogólną organizację tych formatów danych.

Tabela 5.5. Przykłady formatów używanych w zbiorach danych instrukcji

Nazwa	Format JSONL
Alpaca	<pre>{"instruction": "...", "input": "...", "output": "..."} {"instruction": "...", "output": "..."} </pre>
ShareGPT	<pre>{"conversations": [{"from": "...", "value": "..."}, ...]} </pre>
OpenAI	<pre>{"conversations": [{"role": "...", "content": "..."}, ...]} </pre>
OASST	<pre>{"INSTRUCTION": "...", "RESPONSE": "..."} </pre>
Tekst nieprzetworzony	<pre>{"text": "..."} </pre>

W tym miejscu trzeba dodać, że w formacie Alpaca klucz "input" jest opcjonalny. Zawartość tego klucza jest dołączana do zawartości klucza "instruction", o ile taki istnieje. W tabeli zamieściliśmy również format danych „tekst nieprzetworzony”, aby pokazać, że nadzorowane dostrajanie modelu nie różni się zbytnio od wstępnego trenowania. Jeżeli zdecydujesz się na ponowne trenowanie modelu na podstawie tekstu nieprzetworzonego, taki rodzaj dostrajania jest nazwany „ciągłym wstępnym trenowaniem”.

Utworzony we wcześniejszej części rozdziału zbiór danych zawiera dwie kolumny ("instruction" i "output") oraz odpowiada formatowi Alpaca. Ten format jest wystarczający dla pojedynczej instrukcji, która składa się z jednego pytania i jednej odpowiedzi na nie. Jeżeli chcesz przetwarzać rozmowy (wiele pytań i odpowiedzi), wówczas lepiej skorzystać z formatów typu ShareGPT lub OpenAI. Ponieważ poszczególne komunikaty są przechowywane w postaci słownika na liście, mogą przedstawiać dowolnie długie rozmowy w każdej próbce.

Wybór między jedną rozmową lub wieloma ma bezpośredni wpływ na typ pamięci masowej oraz zależy od końcowego przypadku użycia.

Szablony czatu

Gdy pary instrukcji są przetwarzane z formatu zbioru danych, chcemy nadać im strukturę odpowiadającą szablowni czatu. Tego rodzaju szablony oferują ujednolicone sposoby przedstawiania pytań i odpowiedzi modelowi.

Ogólnie rzecz biorąc, zawierają tokeny specjalne, które są przeznaczone do wskazywania początku i końca komunikatu lub jego autora. Ponieważ modele bazowe nie zostały zaprojektowane pod kątem stosowania się do instrukcji, nie mają szablonu czatu. Zatem podczas dostrajania modelu bazowego można wybrać dowolny szablon czatu. Jeżeli chcesz dostroić model instrukcji (niezalecane podejście), musisz skorzystać z tego samego szablonu, ponieważ w przeciwnym razie wydajność działania może spaść.

Podobnie jak w przypadku formatów zbioru danych, istnieją różne rodzaje szablonów czatu: ChatML, Llama 3, Mistral i wiele innych. W społeczności skupionej wokół rozwiązań otwartoźródłowych dużą popularność zyskał szablon ChatML (pierwotnie pochodzący z OpenAI). Dodaje on dwa tokeny specjalne — `<|im_start|>` i `<|im_end|>` — wskazujące autora wypowiedzi. Aby pokazać przykład, w tabeli 5.6 zamieściliśmy wynik otrzymany po zastosowaniu szablonu ChatML dla pary instrukcji z tabeli 5.1.

Tabela 5.6. Próbką z tabeli 5.1 po zastosowaniu szablonu czatu ChatML

```
<|im_start|>system
Jesteś pomocnym asystentem, który wszystko dokładnie wyjaśnia. Odpowiadaj tak,
jakbyś rozmawiał z pięciolatką. <|im_end|>
<|im_start|>user
Pojęcia: budynek, sklep, miasto
Utwórz zdanie, które zawiera wszystkie wymienione słowa. <|im_end|>
<|im_start|>assistant
W naszym małym mieście znajduje się duży budynek mieszczący sklep, w którym
klienci kupują ulubione zabawki i słodycze. <|im_end|>
```

Jak widać, wciąż mamy trzy oddzielne części: system, użytkownik (user) i asystent (assistant). Każda z tych części rozpoczyna się tokenem `<|im_start|>` i kończy tokenem `<|im_end|>`. Bieżący użytkownik jest identyfikowany za pomocą ciągu tekstowego (np. system) zamiast tokena specjalnego. To jest dokładny ciąg tekstowy, który został tokenizowany i model używa go jako danych wejściowych podczas dostrajania.

Jednak w trakcie wnioskowania nie można podać oczekiwanej odpowiedzi. Wówczas dostarczane są części systemu i użytkownika (rysunek 5.6) i prosimy model o udzielenie odpowiedzi poprzez dodanie `<|im_start|>asystent\n`.

Ponieważ model został dostrojony za pomocą podanego szablonu, doskonale wie, że następne tokeny powinny zawierać odpowiedź, która jest związana z instrukcją użytkownika i promptem systemowym. W taki właśnie sposób dostrajane modele nabywają umiejętność stosowania się do instrukcji.

W przypadku szablonów czatu powszechnie występującym problemem jest to, że każdy biały znak i znak nowego wiersza są niezwykle ważne. Dodawanie lub usuwanie tych znaków może prowadzić do nieprawidłowej tokenizacji, co niekorzystnie wpłynie na wydajność działania modelu. Dlatego zaleca się używanie niezawodnych szablonów, takich jak Jinja, które zostały zaimplementowane w bibliotece Transformers. W tabeli 5.7 przedstawiliśmy kilka przykładów takich szablonów, w tym szablon Alpaca (to zarówno format zbioru danych instrukcji, jak i szablon czatu).

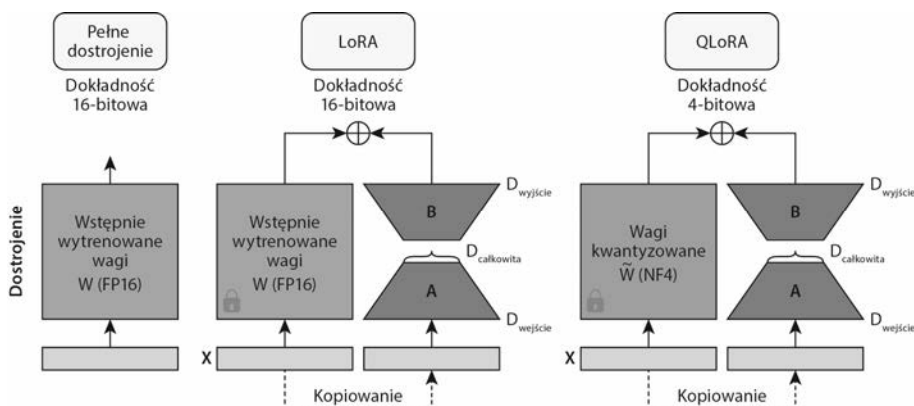
Tabela 5.7. Przykłady najczęściej spotykanych szablonów czatu

Nazwa	Szablon Jinja
Alpaca	### Instruction: Jakie miasto jest stolicą Francji? ### Response: Stolicą Francji jest Paryż.<EOS>
ChatML	< im_start >user Jakie miasto jest stolicą Francji?< im_end > < im_start >assistant Stolicą Francji jest Paryż.< im_end >
Llama 3	< begin_of_text >< start_header_id >user< end_header_id > Jakie miasto jest stolicą Francji?< eot_id >< start_header_id >assistant< end_header_id > Stolicą Francji jest Paryż.< eot_id >
Phi-3	< user > Jakie miasto jest stolicą Francji?< end > < assistant > Stolicą Francji jest Paryż.< end >
Gemma	<bos><start_of_turn>user Jakie miasto jest stolicą Francji?<end_of_turn> <start_of_turn>model Stolicą Francji jest Paryż.<end_of_turn>

W szablonach Jinja zostały zaimplementowane pętle i konstrukcje warunkowe, dzięki którym ten sam szablon może być wykorzystywany zarówno podczas trenowania, jak i wnioskowania (add_generation_prompt).

Techniki Parameter-Efficient Fine-Tuning (PEFT)

Wprawdzie w literaturze można znaleźć wiele technik, ale nadzorowane dostrajanie modelu koncentruje się na trzech z nich, a są to: pełne dostrajanie, LoRA i QLoRA (rysunek 5.9). Omówimy je po kolei oraz przedstawimy ich wady i zalety w zależności od przypadków użycia.



Rysunek 5.9. Architekturalne różnice między trzema najważniejszymi technikami nadzorowanego dostrajania modelu na poziomie modelu

Pełne dostrajanie

Pełne dostrajanie odwołuje się do najprostszej techniki nadzorowanego dostrajania modelu i obejmuje ponowne trenowanie każdego parametru w modelu bazowym. Podobnie jak wstępne trenowanie, nadzorowane dostrajanie modelu używa prognozowania następnego tokena jako celu uczenia. Zatem wcześniej omówiona struktura zbioru danych może być postrzegana jako największa różnica między ciągłym wstępnym trenowaniem i pełnym dostrajaniem.

Ta metoda często zapewnia najlepsze wyniki, przy czym wymaga znacznych zasobów obliczeniowych. Poziom użycia pamięci zależy od wielu czynników, między innymi wielkości modelu, technik trenowania oraz metod optymalizacji. W najprostszej postaci, w przypadku użycia jednej karty graficznej, ilość niezbędnej pamięci można oszacować za pomocą następującego wzoru:

$$\text{pamięć} = \text{parametry} + \text{gradienty} + \text{stany optymalizatora} + \text{aktywacje}$$

W przypadku prostej konfiguracji obejmującej dokładność **32-bitowych liczb zmiennoprzecinkowych (FP32)** można dokonać następującego oszacowania:

- **Parametry.** Są to możliwe do nauczenia wagi i tendencyjność w sieci neuronowej. W dużym modelu językowym zwykle będą to wagi w mechanizmie uwagi, warstwy przejścia do przodu i warstwy osadzania. Koszt to 4 bajty (FP32) lub 2 bajty (FP16/BF16) na parametr.
- **Gradienty.** Są to częściowe pochodne funkcji straty w odniesieniu do poszczególnych parametrów modelu. Wskazują, w jakim stopniu parametry powinny być dostosowane, aby zminimalizować stratę. Podczas trenowania gradienty są obliczane dla poszczególnych parametrów za pomocą propagacji wstecznej i służą do uaktualniania parametrów modelu. Koszt to 4 bajty na parametr.
- **Stany optymalizatora.** Są to wartości dodatkowe obsługiwane przez algorytmy optymalizacji takie jak Adam i AdamW. Zwykle obejmują średnie ruchome poprzednich gradientów i kwadraty poprzednich gradientów dla poszczególnych parametrów. Pomagają w dostosowaniu tempa uczenia

poszczególnych parametrów, aby skuteczniej poruszać się wśród strat. Na przykład algorytm Adam obsługuje dwie wartości dodatkowe (impet i sprzeczność) dla każdego parametru. Koszt to 8 bajtów na parametr (w przypadku optymalizatora Adam).

- **Aktywacje.** Są to pośrednie dane wyjściowe poszczególnych warstw w sieci neuronowej podczas jej przebiegu do przodu. W przypadku modeli opartych na transformerach obejmuje to dane wyjściowe mechanizmów uwagi, warstw przejścia do przodu i warstw normalizacji. W trakcie przebiegu do przodu aktywacje muszą znajdować się w pamięci, aby w przebiegu wstecznym można było obliczyć gradienty, o ile nie zostaną użyte techniki takie jak punkty kontrolne aktywacji. Koszt jest zmienny, często nieistotny w przypadku małych partii.

Tym samym otrzymujemy punkt bazowy o wielkości 16 bajtów na parametr, co przekłada się na 112 GB pamięci VRAM dla modelu 7B oraz 1120 GB dla modelu 70B. Jednak te wartości są często zaniżone, ponieważ nie uwzględniają dodatkowej pamięci niezbędnej dla aktywacji, buforów tymczasowych i obciążenia powodowanego przez różne techniki trenowania.

Istnieje możliwość zastosowania wielu technik w celu zmniejszenia poziomu użycia pamięci podczas dostrajania dużego modelu językowego. Równoległość modelu powoduje rozłożenie obciążenia między wiele kart graficznych, choć to również wiąże się z pewnym obciążeniem. Dzięki akumulacji gradientu można efektywnie przetwarzać większe partie, co nie będzie powodowało proporcjonalnego zwiększenia poziomu użycia pamięci. Optymalizatory przeznaczone do poprawy efektywności użycia pamięci, takie jak 8-bitowy Adam, mogą zmniejszyć wielkość stanów optymalizatora. Punkty kontrolne aktywacji prowadzą do zmniejszenia niezbędnej mocy obliczeniowej, ale kosztem większego zapotrzebowania na pamięć, co wiąże się z ponownym obliczaniem określonych aktywacji. Łącząc te techniki, można znacznie ograniczyć ilość wymaganej pamięci. Na przykład połączenie zmiennej dokładności i równoległości modelu pozwala obniżyć koszt do mniej więcej 14 – 15 bajtów na parametr w porównaniu do 16 bajtów w punkcie wyjścia. Jednak wymagania w zakresie pamięci pozostają znaczne w przypadku dużych modeli językowych, nawet po zastosowaniu wymienionych technik optymalizacji.

Trzeba również dodać, że pełne dostrajanie bezpośrednio modyfikuje wstępnie wytrenowane wagi, co z natury jest działaniem destrukcyjnym. Jeżeli trenowanie nie działa zgodnie z oczekiwaniami, może prowadzić do wymazania poprzedniej wiedzy i umiejętności — jest to zjawisko określane mianem „katastrofalnego zapomnienia”. Z tym samym zjawiskiem możemy mieć do czynienia podczas ciągłego wstępnego trenowania, więc omówione techniki są, ogólnie rzecz biorąc, znacznie trudniejsze w użyciu. Ze względu na tę dodatkową złożoność i związane z nią wysokie wymagania obliczeniowe podczas tworzenia modeli związanych z zadaniem lub dziedzina często są preferowane techniki PEFT zamiast pełnego dostrajania.

LoRA

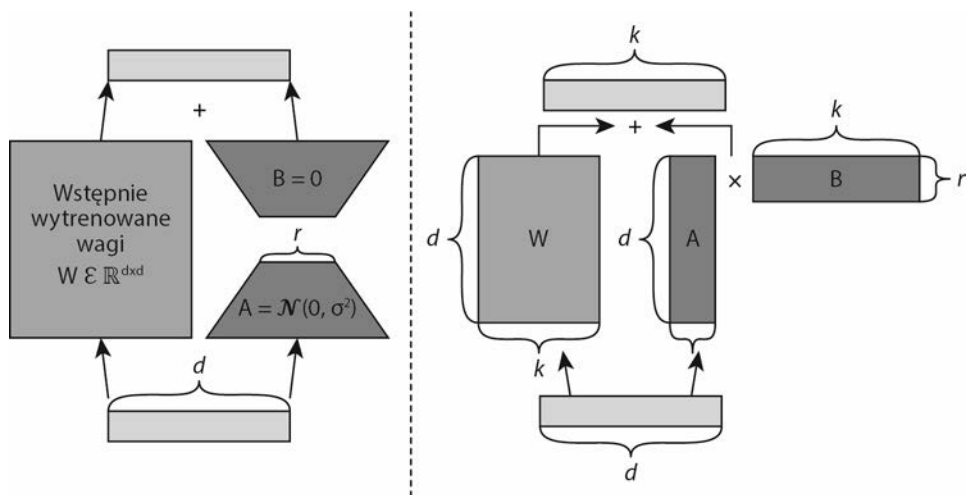
LoRA jest techniką PEFT przeznaczoną do dostrajania dużych modeli językowych. Została opracowana w celu rozwiązywania problemów obliczeniowych, które są związane z ogromnymi sieciami neuronowymi. Bardzo szybko stała się podstawową techniką używaną w trakcie dostrajania dużych modeli językowych.

Głównym zadaniem techniki LoRA jest umożliwienie dostrajania dużych modeli językowych z wykorzystaniem znacznie mniejszej ilości zasobów obliczeniowych. Jest to wykonalne dzięki wprowadzeniu możliwych do trenowania macierzy niskiej rangi, które modyfikują sposób działania modelu, nie zmieniając przy tym jego pierwotnych parametrów. Do najważniejszych zalet techniki LoRA zaliczamy:

- znacznie niższe użycie pamięci podczas trenowania;
- szybszy proces dostrajania;
- zachowanie wstępnie wytrenowanych wag modelu (działanie niedestrukcyjne);
- możliwość efektywnego przechodzenia między zadaniami poprzez zmianę wag LoRA.

Te zalety powodują, że LoRA okazuje się szczególnie atrakcyjna dla badaczy i programistów, którzy pracują w środowiskach o ograniczonych zasobach obliczeniowych, co demokratyzuje proces dostrajania dużego modelu językowego.

Działanie techniki LoRA polega na wykorzystaniu rozkładu macierzy niskiej rangi w celu efektywnego uaktualnienia wag modelu. Zamiast bezpośrednio modyfikować pierwotne wagi macierzy W , LoRA wprowadza dwie mniejsze macierze, A i B , które razem tworzą macierz niskiej rangi przeznaczoną do uaktualnienia W (rysunek 5.10).



Rysunek 5.10. Technika LoRA dodaje dwie możliwe do wytrenowania macierze A i B oraz zachowuje wstępnie wytrenowane wagi W

Matematycznie można to przedstawić za pomocą następującego wzoru:

$$W' = W + BA$$

W oznacza pierwotną macierz wag, B i A to macierze LoRA, W' natomiast to efektywna macierz wag używana podczas wnioskowania.

Wielkość macierzy A i B została wybrana w taki sposób, aby ich iloczyn miał kształt odpowiadający W , ale o znacznie niższej randze. Ta ranga, zwykle oznaczana jako r , jest w technice LoRA hiperparametrem o kluczowym znaczeniu. W trakcie trenowania

pierwotne wagi W pozostają nienaruszone, uaktualniane są tylko macierze A i B . Takie podejście znacznie zmniejsza liczbę parametrów możliwych do wytrenowania, co z kolei prowadzi do znacznego zredukowania ilości wymaganej pamięci oraz skraca czas trenowania.

Aby efektywnie zaimplementować technikę LoRA, należy wybrać właściwe hiperparametry i moduły docelowe. LoRA dostarcza dwa hiperparametry:

- **Ranga (r)**. Określa wielkość macierzy LoRA. Powszechnie stosowanym punktem wyjścia jest $r = 8$, ale wartości do 256 w pewnych przypadkach prowadzą do dobrych wyników. Większa ranga może przechwytywać bardziej zróżnicowane zadania, choć z drugiej strony może prowadzić do przetrenowania.
- **Alfa (α)**. Współczynnik skalowania, który jest stosowany dla uaktualnienia LoRA. W praktyce przeprowadza się uaktualnienie zamrożonych wag W przez współczynnik α/r . Dlatego powszechnie przypisuje się α dwukrotną wartość r , co w praktyce oznacza zastosowanie współczynnika skalowania 2 dla uaktualnienia LoRA. W przypadkach przetrenowania lub niewystarczającego wytrenowania można eksperymentować z innymi wartościami tego współczynnika.

Ponadto można dodać warstwę z wartością dropout, aby uniknąć przetrenowania. Ta wartość zwykle mieści się w przedziale od 0 do 0,1 i stanowi opcjonalny współczynnik regulacji, który nieco spowalnia szybkość uczenia.

Technikę LoRA można stosować dla różnych fragmentów architektury modelu. Początkowo LoRA koncentrowała się na modyfikowaniu mechanizmu uwagi, w szczególności macierzy **zapytania** (ang. *query*), **Q**, i **wartości** (ang. *value*), **V**, w warstwie transformera. Jednak eksperymenty dowiodły, że można uzyskać znaczne korzyści wynikające z zastosowania techniki LoRA również w innych kluczowych komponentach modelu. Do tych dodatkowych modułów docelowych zaliczamy:

- macierze **klucza** (ang. *key*), **K**, w warstwach uwagi;
- warstwy projekcji danych wyjściowych (oznaczane jako O) w mechanizmach uwagi;
- bloki przejścia do przodu lub **MLP** (ang. *multi-layer perceptron*) między warstwami uwagi;
- liniowe warstwy danych wyjściowych.

Trzeba pamiętać, że zwiększenie liczby modułów wykorzystujących technikę LoRA prowadzi także do zwiększenia liczby parametrów możliwych do wytrenowania i tym samym do zwiększenia wymagań dotyczących pamięci.

Za pomocą techniki LoRA można dostroić model zawierający 7B parametrów, używając tylko jednej karty graficznej wyposażonej w 14 – 18 GB pamięci VRAM, w zależności od konkretnej konfiguracji. Są to dużo mniejsze wymagania w porównaniu do pełnego dostrojenia, które zazwyczaj wymaga wielu potężnych kart graficznych. W zakresie parametrów możliwych do wytrenowania technika LoRA znacznie zmniejsza ich liczbę w porównaniu do pełnego dostrojenia. Na przykład nawet gdy celujemy w moduły o randze 16, model Llama 3 8B będzie miał jedynie 42 miliony możliwych do wytrenowania parametrów LoRA spośród 8 miliardów, co daje 0,5196% parametrów modelu.

Pod względem jakości LoRA może osiągnąć porównywalne lub niekiedy nawet lepsze wyniki niż pełne dostrajanie. Wiele zbiorów wag LoRA może być łączonych na potrzeby różnych zadań lub dziedzin, co umożliwia elastyczne wdrożenia i zmianę zadań bez konieczności ponownego trenowania. Różne projekty specjalizują się w udostępnianiu wielu technik LoRA. Przykładem może być tutaj LoRAX. To również jest funkcjonalność obsługiwana przez technologie serwisu Hugging Face: **Text Generation Inference (TGI)** i **Nvidia Inference Microservices (NIM)**.

QLoRA

Wprowadzona przez Dettmersa i innych QLoRA jest metodą dostrajania dużych modeli językowych, która radzi sobie z wyzwaniami związanymi z wysokim kosztem zasobów obliczeniowych. Dzięki połączeniu technik optymalizacji oraz technik LoRA i QLoRA programiści mogą dostrajać modele za pomocą względnie małych i powszechnie dostępnych kart graficznych.

Istota podejścia techniki QLoRA polega na kwantyzacji parametrów modelu na postać niestandardowego typu danych **NF4 (4-bit NormalFloat)**, co pozwala znacząco zmniejszyć poziom użycia pamięci. W technice QLoRA, podobnie jak w przypadku techniki LoRA, zamiast uaktualniać wszystkie parametry modelu podczas dostrajania, zostają wprowadzone małe i możliwe do wytrenowania macierze rang (adaptery) w określonych warstwach modelu. Podczas trenowania uaktualniane są jedynie te adaptery, natomiast pierwotne wagi modelu pozostają nienaruszone. Aby jeszcze bardziej zmniejszyć poziom użycia pamięci, QLoRA korzysta z podwójnej kwantyzacji, czyli poddaje kwantyzacji same stałe wykorzystywane przez proces kwantyzacji. Dodatkowo korzysta z optymalizatorów stronicowanych w celu zarządzania nagłymi skokami użycia pamięci podczas trenowania, co umożliwia funkcjonalność pamięci zunifikowanej oferowana przez firmę Nvidia.

QLoRA pozwala zaoszczędzić sporo pamięci w porównaniu do techniki LoRA oraz zmniejszyć nawet o 75% nagłe skoki poziomu użycia pamięci. Na przykład dla modelu 7B technika QLoRA może ograniczyć poziom użycia pamięci z 14 GB do 9,1 GB podczas inicjalizacji, co oznacza spadek o 35%. W trakcie dostrajania spadek użycia pamięci sięga 40%, z 15,6 GB do 9,3 GB dla techniki QLoRA. Jednak ta efektywność pod względem użycia pamięci ma swój koszt, którym jest wydłużenie czasu trenowania — QLoRA jest mniej więcej o 30% wolniejsza niż LoRA. Pod względem wydajności działania modelu QLoRA wykazuje jedynie niewielkie różnice w porównaniu do LoRA.

Podsumowując: QLoRA jest szczególnie użyteczna, gdy największym problemem jest ograniczona ilość pamięci, na przykład podczas pracy z dużymi modelami w komputerze z małą ilością pamięci graficznej. Jeżeli jednak szybkość trenowania ma znaczenie krytyczne i jest dostępna wystarczająca ilość pamięci, to LoRA może okazać się preferowanym wyborem.

Wybór między technikami QLoRA i LoRA powinien odbywać się na podstawie konkretnych wymagań projektu, dostępnego sprzętu komputerowego oraz potrzeby równoważenia poziomu użycia pamięci, szybkości trenowania i wydajności działania modelu.

Trenowanie parametrów

Podczas dostrajania dużych modeli językowych wiele hiperparametrów będzie miało wpływ na proces trenowania oraz duży wpływ na konwergencję, generalizację i na ogólną efektywność modelu.

Harmonogram i tempo uczenia

Tempo uczenia jest najważniejszym hiperparametrem. Określa, jak bardzo parametry modelu będą uaktualniane w trakcie trenowania. Jego wartość jest zróżnicowana, zwykle od bardzo małej, rzędu $1e-6$, do większej, na poziomie $1e-3$. Powszechnie przyjmowanym punktem wyjścia dla modeli transformera jest wartość w okolicach $1e-5$. Jeżeli tempo uczenia jest zbyt wolne, proces trenowania będzie odbywał się bardzo powoli i może utknąć w nieoptymalnych rozwiązaniach. Jeśli zaś będzie za szybkie, trenowanie stanie się niestabilne bądź zbyt zróżnicowane, co będzie prowadzić do kiepskiej wydajności działania modelu. Często korzystne jest eksperymentowanie z różnymi wartościami tempa uczenia, aby znaleźć optymalną wartość dla konkretnego zadania i modelu.

Harmonogram tempa uczenia dostosowuje jego szybkość w trakcie całego procesu trenowania. Na początku tempo przeważnie jest szybsze, aby umożliwić szybki postęp na początku procesu, a następnie powoli się zmniejsza na kolejnych etapach, by precyzyjnie dostroić model. Najczęściej spotykane typy harmonogramów to liniowy i cosinusowy. Harmonogram liniowy powoli zmniejsza tempo uczenia na przestrzeni czasu, natomiast cosinusowy jest zgodny z krzywą cosinusa, czyli najpierw zmniejsza się znacznie wolniej, a potem przyspiesza. Na przykład możesz rozpocząć od tempa uczenia na poziomie $3e-4$, a później zmniejszać je do $1e-7$ w trakcie trenowania. Konkretnie wartości i harmonogram ich zmniejszania będą zależały od modelu i zbioru danych. Powszechne podejście polega na zastosowaniu etapu rozgrzewki (np. 5% całkowitych kroków), w którym tempo uczenia rośnie od 0 do wartości początkowej, a później przez pozostałe 95% kroków powoli spada. Takie podejście pomaga stabilizować wczesne trenowanie i umożliwia stosowanie lepiej dopasowanych uaktualnień w miarę zbliżności modelu. Ogólnie rzecz biorąc, harmonogramy liniowy i cosinusowy zapewniają tę samą wydajność działania.

Wielkość partii

Wielkość partii określa liczbę próbek, które zostaną przetworzone, zanim nastąpi uaktualnienie wag modelu. Typowa wielkość partii podczas dostrajania dużego modelu językowego waha się w przedziale od 1 do 32, przy czym najczęściej stosowane są wartości 1, 2, 4, 8 lub 16. Ogólnie rzecz biorąc, większa partia prowadzi do stabilniejszych szacunków gradientu oraz może poprawić szybkość trenowania, ponieważ daje lepsze przybliżenie faktycznego gradientu w całym zbiorze danych.

Jednak większa partia wymaga również więcej pamięci, co może okazać się czynnikiem ograniczającym w przypadku kart graficznych wyposażonych w mniejszą ilość pamięci VRAM. Na przykład partia o wielkości 16 może sprawdzać się świetnie w przypadku dobrej karty graficznej z 24 GB pamięci, podczas gdy słabsze karty graficzne z 8 GB pamięci mogą obsługiwać jedynie partie o wielkości 2 lub 4.

Aby poradzić sobie z ograniczeniami dotyczącymi ilości pamięci i nadal czerpać korzyści z większych partii, można skorzystać z techniki nazywanej akumulacją gradientu. Jej działanie polega na wykonaniu wielu przejść do przodu i do tyłu z użyciem mniejszych minipartii oraz akumulowaniu gradientów w trakcie tych kroków, przed przeprowadzeniem pojedynczej operacji uaktualnienia parametrów modelu. To podejście jest szczególnie użyteczne w pracy z ogromnymi modelami bądź z ograniczoną ilością pamięci karty graficznej. Na przykład jeśli chcesz osiągnąć efektywną wielkość partii wynoszącą 32, ale karta graficzna jest w stanie obsłużyć jednorazowo tylko 8 próbek, możesz zdefiniować cztery kroki akumulacji gradientu. Wówczas będziesz przetwarzać 4 minipartie o wielkości 8 próbek każda, akumulować gradienty, a następnie uaktualniać model, jakby jednocześnie zostały przetworzone 32 próbki.

Liczba kroków akumulacji gradientu zwykle wynosi od 1 (brak akumulacji) do 8 lub 16, w zależności od oczekiwanej efektywnej wielkości partii oraz od dostępnych zasobów obliczeniowych. Podczas wyboru liczby kroków należy pamiętać o kompromisie między szybkością uczenia i poziomem użycia pamięci. Więcej kroków akumulacji umożliwia stosowanie efektywnie większych partii, a zarazem wydłuża czas każdego uaktualnienia. Oto wzór na obliczenie efektywnej wielkości partii:

$$\text{efektywna wielkość partii} = \text{wielkość partii} \cdot \text{liczba kart graficznych} \cdot \text{liczba kroków akumulacji gradientu}$$

Na przykład jeśli używasz dwóch kart graficznych, a każda przetwarza partię 4 próbek i stosuje 4 kroki akumulacji gradientu, to efektywna wielkość partii będzie wynosiła $4 \cdot 2 \cdot 4 = 32$ próbki.

Maksymalna długość sekwencji i pakowanie

Maksymalna długość sekwencji określa najdłuższe dane wejściowe, jakie model będzie w stanie przetworzyć. Zwykle wynosi ona od 512 do 4096 tokenów, ale może sięgać także 1 280 000 lub więcej tokenów, w zależności od zadania i dostępnej ilości pamięci karty graficznej. Na przykład wielkość maksymalna 2048 tokenów jest powszechnie stosowana w wielu zadaniach generowania wykonywanych przez modele językowe, a aplikacje wykorzystujące technikę RAG mogą używać 8192 lub więcej tokenów. Podczas przetwarzania danych wejściowych sekwencje dłuższe niż zdefiniowany limit zostaną skrócone, co oznacza usunięcie nadmiarowych tokenów. Operacja skracania może odbywać się na początku (skraccanie lewe) lub na końcu (skraccanie prawe) sekwencji. Na przykład jeśli maksymalna długość sekwencji wynosi 1024 tokeny, a dane wejściowe składają się z 1500 tokenów, to 476 tokenów zostanie usuniętych. Ten parametr ma bezpośredni wpływ na wielkość partii i poziom użycia pamięci. Zatem partia o wielkości 12 i maksymalnej długości sekwencji 1024 będzie zawierać 12 288 tokenów ($12 \cdot 1024$), podczas gdy ta sama partia, ale o maksymalnej długości sekwencji 512 będzie zawierać jedynie 6144 tokeny. Bardzo ważne jest równoważenie tego parametru z możliwościami karty graficznej i naturą danych używanych do uczenia, aby zoptymalizować wydajność działania i poziom użycia zasobów.

Pakowanie maksymalizuje poziom wykorzystania poszczególnych partii trenowania. Zamiast przypisywać tylko jedną próbkę dla partii, poprzez pakowanie można połączyć wiele mniejszych próbek w pojedynczą partię, a tym samym efektywnie zwiększyć ilość danych, które są przetwarzane podczas danej iteracji. Na przykład jeśli maksymalna

długość sekwencji wynosi 1024 tokeny, a wiele próbek składa się jedynie z 200 – 300 tokenów, to pakowanie umożliwia zmieszczenie 3 – 4 próbek w każdej partii. Takie podejście może znacznie poprawić efektywność trenowania, zwłaszcza podczas pracy ze zbiorami danych, które zawierają wiele krótkich sekwencji. Jednak pakowanie wymaga starannej implementacji, aby uwaga modelu nie przechodziła między pakowanymi próbkami. Zazwyczaj można to osiągnąć dzięki zastosowaniu masek uwagi, które uniemożliwiają modelowi używanie w tej samej pakowanej sekwencji tokenów pochodzących z różnych próbek.

Liczba epok

Liczba epok to kolejny ważny parametr, reprezentujący liczbę ukończonych przejść przez cały zbiór danych. W przypadku dostrajania modelu typowy zakres wynosi od 1 do 10 epok, a wiele zakończonych powodzeniem przebiegów obejmuje od 2 do 5 epok. Optymalna liczba będzie zależała od różnych czynników, takich jak złożoność zadania, wielkość zbioru danych oraz architektura modelu. Więcej epok umożliwia modelowi dostosowanie uczenia, co może przynieść poprawę wydajności działania. Jednak trzeba pamiętać o ważnym kompromisie: zbyt mała liczba epok może prowadzić do niewystarczającego wytrenowania, zbyt duża zaś może spowodować przetrenowanie. Na przykład duży model językowy dostrojony za pomocą małego zbioru danych może potrzebować jedynie od 1 do 3 epok, a w przypadku mniejszego modelu dostrojonego z użyciem większego zbioru danych korzystne może być od 5 do 10 epok. Warto monitorować wydajność działania weryfikacji podczas trenowania i wcześniej zakończyć operację, jeśli wydajność działania modelu osiągnie stan stagnacji bądź się pogorszy. Takie podejście pomaga w dynamicznym ustaleniu optymalnej liczby epok i zapobiega przetrenowaniu.

Optymalizatory

Optymalizatory dostrajają parametry modelu, aby minimalizować funkcję straty. W przypadku dostrajania dużego modelu językowego szczególnie zalecanym algorytmem jest AdamW (ang. *Adaptive Moment Estimation with Weight Decay*), zwłaszcza jego 8-bitowa wersja. Jego wydajność działania jest porównywalna z wersją 32-bitową, a jednocześnie używa mniej pamięci graficznej (ale nie zwiększa szybkości trenowania). AdamW łączy adaptacyjne tempo uczenia z regulacją zaniku wagi, często prowadząc do lepszej stabilności uczenia i wydajności działania modelu.

W scenariuszach z dużymi ograniczeniami pod względem ilości pamięci algorytm AdaFactor stanowi alternatywę opracowaną pod kątem efektywności poziomu użycia pamięci. Ten algorytm sprawdza się wyjątkowo dobrze bez wyraźnego określania tempa uczenia i jest szczególnie użyteczny w środowiskach charakteryzujących się ograniczonymi zasobami. Jednak nie zawsze jego wydajność będzie odpowiadała wydajności algorytmu AdamW. W przypadku wyjątkowo ogromnych modeli lub naprawdę ograniczonej ilości pamięci graficznej stronicowane wersje optymalizatorów, np. AdamW 8-bit, mogą jeszcze bardziej zmniejszyć poziom użycia pamięci poprzez przekazywanie danych do pamięci operacyjnej RAM. Jeżeli pamięć pozwala, a maksymalna wydajność działania jest priorytetem, wówczas najlepszym wyborem może się okazać niestosujący kwantyzacji optymalizator `adamw_torch`.

Zanikanie wagi

Zanikanie wagi działa poprzez nakładanie kary na ogromne wagi do funkcji straty, aby w ten sposób zachęcić model do uczenia się prostszych i bardziej ogólnych cech. Dzięki temu model unika dużej zależności od pojedynczej cechy wejściowej, co może poprawić wydajność jego działania dla danych, które są przetwarzane po raz pierwszy. Ten parametr z reguły ma wartość w przedziale od 0,01 do 0,1, przy czym 0,01 to często stosowany punkt wyjścia. Na przykład jeśli używasz optymalizatora AdamW, możesz na początku przypisać parametrowi zanikania wagi wartość 0,01.

Wprawdzie ten parametr może być korzystny, ale przypisanie mu zbyt dużej wartości może zakłócać proces uczenia, ponieważ model będzie miał trudności w wychwytywaniu ważnych wzorców w danych. Z kolei zbyt mała wartość może nie zapewniać wystarczającej regulacji uczenia. Wartość optymalna często zależy od architektury konkretnego modelu i zbioru danych, więc, ogólnie rzecz biorąc, dobrą praktyką jest eksperymentowanie z różnymi wartościami.

Punkt kontrolny gradientu

Jest to technika zmniejszająca poziom użycia pamięci w trakcie uczenia, ponieważ przechowuje jedynie podzbiór aktywacji pośrednich, które zostały wygenerowane podczas przejścia do przodu. W standardowych procedurach uczenia wszystkie aktywacje pośrednie pozostają w pamięci, aby tym samym ułatwić obliczanie gradientu w trakcie przejścia wstecznego. Jednak w przypadku niezwykle mocno zagnieżdżonych sieci, na przykład takich jak duże modele językowe, takie podejście bardzo szybko staje się niepraktyczne ze względu na ograniczenia sprzętowe, zwłaszcza dotyczące kart graficznych z małą ilością pamięci VRAM.

Punkt kontrolny gradientu pomaga poradzić sobie z tym problemem dzięki selektywemu zapisywaniu aktywacji w określonych warstwach sieci. W przypadku warstw, w których aktywacje nie są zapisywane, będą one ponownie obliczane w trakcie przejścia wstecznego, ponieważ jest to konieczne dla obliczenia gradientu. Takie podejście prowadzi do kompromisu między czasem obliczeń i poziomem użycia pamięci. Wprawdzie powoduje znaczne obniżenie poziomu niezbędnej pamięci, ale jednocześnie prowadzi do wydłużenia ogólnego czasu obliczeń, co wynika z konieczności ponownego obliczenia pewnych aktywacji.

Istnieją jeszcze inne parametry i techniki, ale odgrywają one małą rolę w porównaniu do tutaj omówionych. W następnym podrozdziale na konkretnym przykładzie wyjaśnimy, jak wybrać i dostroić te parametry.

Dostrajanie w praktyce

W tym podrozdziale dostroimy model otwartoźródłowy z wykorzystaniem własnego zbioru danych. Pokażemy przykład, w których zostały zaimplementowane techniki LoRA i QLoRA w celu zapewnienia efektywności. W zależności od używanego sprzętu komputerowego możesz wybrać technikę, która w najlepszym stopniu odpowiada Twojej konfiguracji.

Istnieje wiele efektywnych modeli o otwartych wagach, które można wykorzystać w przypadkach związanych z zadaniem lub dziedziną. Aby wybrać najodpowiedniejszy duży model językowy, należy rozważyć trzy główne parametry:

Licencja. W przypadku niektórych modeli ich licencja zezwala na wykorzystanie tylko w projektach niekomercyjnych, co może stanowić problem, jeśli dostrojenie ma być przeprowadzone na potrzeby firmy. W dziedzinie modeli językowych istnieje wiele licencji niestandardowych, które mogą dotyczyć firm posiadających na przykład określoną liczbę użytkowników.

Budżet. Modele o mniejszej liczbie parametrów (poniżej 10 miliardów) są znacznie tańsze do dostrojenia i wdrożenia na potrzeby wnioskowania niż modele o większej liczbie parametrów. Wynika to z faktu, że takie modele mogą działać na tańszych kartach graficznych i przetwarzają więcej tokenów na sekundę.

Wydajność. Kluczowe znaczenie ma ocena modelu bazowego za pomocą ogólnego przeznaczenia testu wydajności lub, jeszcze lepiej, związanego z zadaniem lub dziedziną, istotnych dla wybranego przypadku użycia. To pomaga zagwarantować, że po dostrojeniu model będzie oferował niezbędne możliwości i oczekiwaną wydajność podczas wykonywania określonych zadań.

W tym rozdziale zdecydowaliśmy się na model Llama 3.1 8B, czyli model otwartoźródłowy udostępniony przez firmę Meta. Ten model został wydany na liberalnej licencji („Llama 3.1 Community License Agreement”), której warunki zezwalają na jego stosowanie w rozwiązaniach komercyjnych. Zawiera 8 miliardów parametrów, więc jest wystarczająco mały, aby mieścić się w pamięci większości kart graficznych. Jednocześnie ma dużą wydajność działania w porównaniu do konkurencyjnych modeli. Te twierdzenia można zweryfikować za pomocą Open LLM Leaderboard oraz innych testów wydajności wymienionych na karcie modelu.

Istnieją wyspecjalizowane narzędzia i biblioteki przeznaczone do dostrajania modeli. W szczególności polecamy następujące:

- **TRL.** Jest to biblioteka utworzona i rozwijana przez serwis Hugging Face, przeznaczona do trenowania dużych modeli językowych za pomocą nadzorowanego dostrajania modelu i uwzględniania preferencji. Ta popularna i niezawodna biblioteka jest najbardziej na czasie pod względem algorytmów. Działa w środowiskach zawierających jedną lub wiele kart graficznych oraz z FSDP i DeepSpeed.
- **Axolotl.** To utworzone przez Wing Lian narzędzie służy do dostrajania dużych modeli językowych poprzez wielokrotne użycie plików konfiguracyjnych w formacie YAML. Opiera się na bibliotece TRL, ale oferuje dodatkową funkcjonalność, np. automatyczne łączenie zbiorów danych, które są przechowywane w różnych formatach. Działa w środowiskach zawierających jedną lub wiele kart graficznych oraz z FSDP i DeepSpeed.
- **Unsloth.** Autorami tej biblioteki są Daniel i Michael Han. Używa ona własnego jądra do przyspieszenia operacji trenowania (2 – 5x) oraz zmniejszenia poziomu użycia pamięci (do 80% mniej). Opiera się na bibliotece TRL oraz dostarcza wiele funkcjonalności, np. automatyczną konwersję modeli na format kwantyzacji GGUF. W chwili powstawania książki działała jedynie w środowiskach wyposażonych w jedną kartę graficzną.

Aby zmaksymalizować efektywność, dostrajanie przeprowadzimy za pomocą biblioteki Unsloth. Zamieszczony tutaj kod jest częścią naszego potoku LLMOps, choć może być również używany w oddzielnym skrypcie. Kod jest przeznaczony do uruchamiania w różnych środowiskach, takich jak na przykład SageMaker, chmura (typu Lambda Labs lub RunPod), Google Colab. Przetestowaliśmy kod z wieloma różnymi kartami graficznymi typu A40, A100 i L4.

Aby zainstalować bibliotekę Unsloth i jej zależności, zalecamy przeprowadzenie bezpośredniej instalacji z poziomu repozytorium GitHuba zawierającego materiały przygotowane dla książki (<https://github.com/PacktPublishing/LLM-Engineers-Handbook>) albo z repozytorium Unsloth (<https://github.com/unslothai/unsloth>). Takie podejście jest zalecane, ponieważ procedura instalacji jest regularnie modyfikowana, aby eliminować potencjalne konflikty z zależnościami.

1. Na początek trzeba uzyskać dostęp do modelu bramy i (opcjonalnie) przekazać dostrojony model do serwisu Hugging Face (<https://huggingface.co/>). Będzie to wymagało zalogowania się do konta w tym serwisie. Jeżeli jeszcze nie masz takiego konta, możesz je utworzyć i umieścić w pliku `.env` swój klucz API (*Settings/Access Tokens/Create new token*):

```
HF_TOKEN = TWÓJ_KLUCZ_API
```

2. Upewnij się, że plik `.env` zawiera również klucz API Comet:

```
COMET_API_KEY = TWÓJ_KLUCZ_API
```

3. Zaimportuj wszystkie niezbędne pakiety:

```
import os
import torch
from trl import SFTTrainer
from datasets import load_dataset, concatenate_datasets
from transformers import TrainingArguments, TextStreamer
from unsloth import FastLanguageModel, is_bfloat16_supported
```

4. Kolejnym krokiem jest wczytanie modelu przeznaczonego do dostrojenia oraz odpowiedniego tokenizera. W omawianym przykładzie użyjemy klasy `FastLanguageModel` biblioteki Unsloth razem z metodą `.from_pretrained()`. Poza nazwą modelu trzeba jeszcze podać maksymalną długość sekwencji (tutaj jest to 2048 tokenów). Argument `load_in_4bit` wskazuje na użycie techniki **QLoRA** lub **LoRA**.

W tym przykładzie zdecydowaliśmy się na technikę LoRA, ponieważ umożliwia szybsze trenowanie oraz oferuje wyższą jakość. Oczywiście możesz bardzo łatwo zastąpić ją techniką QLoRA, jeśli Twój sprzęt nie ma wystarczającej ilości pamięci VRAM.

```
max_seq_length = 2048
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name="meta-llama/Meta-Llama-3.1-8B",
    max_seq_length=max_seq_length,
    load_in_4bit=False,
)
```

5. Po wczytaniu modelu można zdefiniować konfigurację LoRA. W tym przykładzie użyliśmy rangi 32, która okazuje się na tyle duża, aby naśladować styl pisanie i skopiować wiedzę z próbek instrukcji. Jeżeli otrzymane wyniki

będą rozczarowujące, wartość tę można zwiększyć do 64 lub 128. Ponadto parametrowi alfa przypisaliśmy wartość 32, bez opcji dropout i bias, aby przyspieszyć trenowanie. Jako cel wskazujemy wszystkie warstwy liniowe, tym samym maksymalizując jakość procesu dostrajania.

```
model = FastLanguageModel.get_peft_model(
    model,
    r=32,
    lora_alpha=32,
    lora_dropout=0,
    target_modules=["q_proj", "k_proj", "v_proj", "up_proj", "down_proj",
"o_proj", "gate_proj"],
)
```

6. Następnym krokiem jest przygotowanie danych w odpowiednim formacie, które będą użyte do dostrajania. W omawianym przykładzie nie mamy zbyt wielu próbek w zbiorze danych llmtwin (około 3000). To stanowi problem, ponieważ model może nie zostać poprawnie wyuczony za pomocą szablonu czatu. Rozwiązaniem jest upsampling z wykorzystaniem wysokiej jakości ogólnego przeznaczenia zbioru danych o nazwie FineTome. Jest to filtrowana wersja *arcee-ai/The-Tome using the fineweb-edu-classifier*. Zamiast używać 100 000 próbek z tego zbioru danych, zdecydowaliśmy się na wykorzystanie jedynie 10 000 w zbiorze przeznaczonym do trenowania. Następnie połączymy oba te zbiory, aby powstał ostateczny:

```
dataset1 = load_dataset("mlabonne/llmtwin")
dataset2 = load_dataset("mlabonne/FineTome-Alpaca-100k", split="train[:10000]")
dataset = concatenate_datasets([dataset1, dataset2])
```

7. Następnie formatujemy dane z użyciem szablonu czatu. Dla wygody skorzystamy z szablonu Alpaca. Nie wymaga on żadnych dodatkowych tokenów, a tym samym jest mniej podatny na błędy (to może mieć pewien wpływ na wydajność w porównaniu do ChatML). W tym kroku mapujemy wszystkie instrukcje (pytania i odpowiedzi na nie) za pomocą szablonu Alpaca. Na końcu każdej wiadomości ręcznie umieszczamy token **EOS** (ang. *end of sentence*), aby mieć pewność, że model nauczy się go używać. Bez niego model będzie ciągle generować odpowiedzi i nigdy się nie zatrzyma.

alpaca_template = """Poniżej znajduje się instrukcja, która opisuje zadanie.
Wygeneruj odpowiedź, która będzie odpowiednim uzupełnieniem pytania.

```
### Instrukcja:
{}
```

```
### Odpowiedź:
{}
```

```
EOS_TOKEN = tokenizer.eos_token
dataset = dataset.map(format_samples, batched=True,
remove_columns=dataset.column_names)
```

8. Po przygotowaniu zbioru danych można go podzielić na część używaną do trenowania (95%) i do testowania (5%). Ta druga część posłuży do weryfikacji przeprowadzanej podczas trenowania:

```
dataset = dataset.train_test_split(test_size=0.05)
```

9. Model jest teraz gotowy do trenowania. Klasa `SFTTrainer` przechowuje wszystkie hiperparametry na potrzeby trenowania. Ponadto dostarczamy model, tokenizer, konfigurację LoRA i zbiory danych. Stosując się do zaleceń zamieszczonych we wcześniejszej części rozdziału, definiujemy tempo uczenia $3e-4$, harmonogram liniowy i maksymalną długość sekwencji 2048 tokenów. Model trenujemy z użyciem trzech epok, partia ma wielkość 2 i korzystamy z ośmiu kroków akumulacji gradientu (efektywna wielkość partii wynosi wówczas 16). Zdecydowaliśmy się również na optymalizator `adamw_8bit` z opcją `weight_decay` o wartości 0.01. W zależności od dostępnej karty graficznej automatycznie będą używane FP16 lub BF16 dla aktywacji. Informacje dotyczące przebiegu trenowania będą przekazywane do oprogramowania Comet w celu śledzenia eksperymentu.

```
trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
    dataset_text_field="text",
    max_seq_length=max_seq_length,
    dataset_num_proc=2,
    packing=True,
    args=TrainingArguments(
        learning_rate=3e-4,
        lr_scheduler_type="linear",
        per_device_train_batch_size=2,
        gradient_accumulation_steps=8,
        num_train_epochs=3,
        fp16=not is_bfloat16_supported(),
        bf16=is_bfloat16_supported(),
        logging_steps=1,
        optim="adamw_8bit",
        weight_decay=0.01,
        warmup_steps=10,
        output_dir="output",
        report_to="comet_ml",
        seed=0,
    ),
)

trainer.train()
```

Trenowanie tego modelu z wykorzystaniem przygotowanego zbioru danych będzie trwało kilka godzin. Jeżeli użyjesz karty graficznej A100, trenowanie potrwa około 50 minut.

10. Po zakończeniu trenowania model można przetestować za pomocą szybkiego przykładu. Celem nie jest tutaj poprawna ocena dostrojonego modelu, ale raczej upewnienie się, że nie ma oczywistych błędów związanych z tokenizerem lub szablonem czatu.

Na potrzeby szybkiego wnioskowania można skorzystać z wywołania `FastLanguageModel.for_inference()`, które pochodzi z biblioteki Unsloth.

Instrukcję formatujemy bezpośrednio na format Alpaca. Zwróć uwagę na dostarczenie pustej odpowiedzi w celu dołączenia nagłówka asystenta (### Odpowiedź) na końcu instrukcji użytkownika. To wymusza na modelu udzielenie odpowiedzi na pytanie w instrukcji zamiast jego uzupełnienia. Istnieje możliwość użycia strumienia tekstu, aby strumieniować odpowiedź zamiast czekać z jej wyświetleniem aż do chwili jej pełnego wygenerowania.

```
FastLanguageModel.for_inference(model)
message = alpaca_prompt.format("Wygeneruj akapit z wprowadzeniem do nadzorowanego
↳dostrajania modelu.", "")
inputs = tokenizer([message], return_tensors="pt").to("cuda")

text_streamer = TextStreamer(tokenizer)
_ = model.generate(**inputs, streamer=text_streamer, max_new_tokens=256,
↳use_cache=True)
```

11. Oto odpowiedź wygenerowana przez model:

Nadzorowane dostrajanie modelu to metoda stosowana w celu usprawnienia dużego modelu językowego poprzez dostarczenie mu wyselekcjonowanego zestawu danych ↳instrukcji, czyli pytań i odpowiedzi na nie. Proces ten ma na celu dostosowanie odpowiedzi ↳modelu do oczekiwań człowieka, co prowadzi do zwiększenia ich dokładności i trafności. Celem jest sprawić, by model skutecznie odpowiadał na szeroki zakres zapytań, co czyni go cennym narzędziem dla aplikacji takich jak chatboty i wirtualne ↳asystenty.

Jest to poprawna i prawidłowo sformatowana odpowiedź z użyciem szablonu czatu Alpaca.

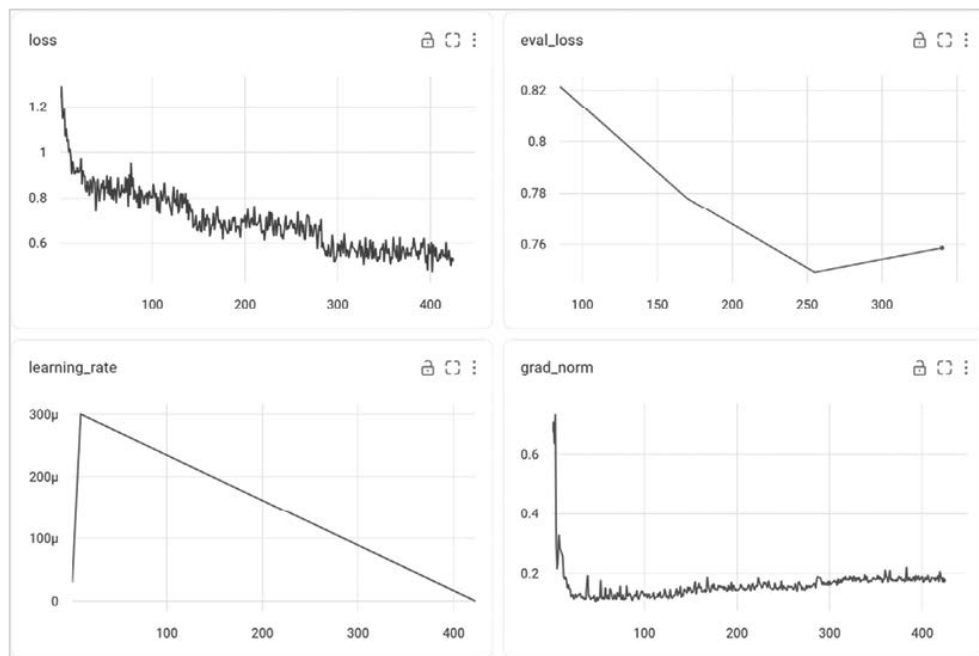
12. Skoro nasz model został poprawnie dostrojony, możemy zapisać go lokalnie i/lub przekazać do serwisu Hugging Face za pomocą przedstawionych funkcji:

```
model.save_pretrained_merged("model", tokenizer, save_method="merged_16bit")
model.push_to_hub_merged("mlabonne/TwinLlama-3.1-8B", tokenizer, save_
↳method="merged_16bit")
```

Gratulacje! W ten sposób dostroiliśmy model bazowy, przeprowadzając całą operację zupełnie od początku. Podczas trenowania możesz uzyskać dostęp do oprogramowania Comet, aby monitorować wskaźniki trenowania, weryfikowania itd. Upewnij się, że wartości tych wskaźników odpowiadają oczekiwanym. Na rysunku 5.11 pokazaliśmy wykresy wygenerowane przez Cometą podczas użycia omówionego wcześniej kodu.

Szczególnie ważne do monitorowania są trzy wskaźniki:

- **Strata podczas trenowania.** Pokazuje, jak model radzi sobie z zadaniem, pod kątem którego jest trenowany. Strata powinna się nieustannie zmniejszać, co będzie wskazywało na poprawę wydajności działania. Można się spodziewać dużego spadku na początku trenowania, a potem długiej stagnacji. Nagłe skoki i ciągle wzrosty wartości straty są sygnałem, że trenowanie podąża w złym kierunku. W takim przypadku należy sprawdzić jakość danych, sprawdzić tokenizer oraz dostroić parametry, takie jak tempo uczenia i wielkość partii. Na rysunku 5.11 wykres „loss” pokazuje istnienie trzech różnych faz odpowiadających użytym epokom.



Rysunek 5.11. Cztery wskaźniki monitorowane w Comecie podczas dostrajania naszego modelu

- **Weryfikacja straty.** Pokazuje stratę zmierzoną dla zbioru weryfikacyjnego, a nie zbioru używanego do trenowania modelu. W przypadku dobrze dostrojonego modelu straty podczas trenowania i weryfikacji zwykle zmniejszają się i ostatecznie stabilizują, a między nimi jest niewielka różnica. Ta różnica powinna być minimalna i jest oczekiwana, ponieważ model zawsze radzi sobie nieco lepiej podczas pracy z danymi wykorzystanymi do trenowania. Jeżeli strata podczas trenowania zmniejsza się, a zaczyna rosnąć podczas testowania, jest to sygnał przetrenowania. Natomiast jeśli oba wykresy są jednakowo płaskie i na wysokim poziomie, wskazuje to na niedostateczne wytrenowanie. Nie istnieją żadne uniwersalne „zalecane zakresy” dla wartości straty, ponieważ zależą one od konkretnego problemu i użytej funkcji straty. Jednak należy zwracać uwagę na zbieżność i stabilność w obu tych wykresach. Na rysunku 5.11 wykres „eval_loss” pokazuje drobny wzrost w punkcie 340. Ten wynik nadal jest akceptowalny, ale może wskazywać, że model zaczyna być przetrenowany.
- **Gradient.** Pokazuje rząd wielkości wektora gradientu podczas trenowania. Ogromny gradient może wskazywać na niestabilne trenowanie, na przykład przetrenowanie, zwłaszcza jeśli towarzyszy temu rozbieżność między stratami w trakcie trenowania i testowania. Natomiast stabilny lub zmniejszający się gradient oznacza, że model zmierza w kierunku lokalnego optimum. Aby poradzić sobie z problemami związanymi z ogromnym gradientem, można zastosować tak zwany *gradient clipping*, czyli technikę, która obejmuje zdefiniowanie maksymalnej wartości progowej dla gradientu i w praktyce ogranicza wielkość uaktualnień parametru.

Interesujące będzie wypróbowanie różnych wartości tempa uczenia oraz wybranie najlepszego modelu na podstawie jak najmniejszej straty. Pamiętaj, że to tylko wstęp do rzeczywistej oceny, którą omówimy w następnym rozdziale.

Podsumowanie

W tym rozdziale przedstawiliśmy ważne aspekty związane z dostrajaniem dużego modelu językowego, zarówno teoretyczne, jak i praktyczne. Przeanalizowaliśmy potok instrukcji danych oraz wyjaśniliśmy, jak należy tworzyć wysokiej jakości zbiory danych, od gromadzenia informacji po ich uzupełnianie. Poszczególne etapy potoku oferują możliwości w zakresie optymalizacji, szczególnie pod kątem oceny jakości, generowania danych i usprawniania. Ten elastyczny potok można dostosować do własnych przypadków użycia poprzez wybór najodpowiedniejszych etapów i technik.

Omówiony framework zastosowaliśmy dla rzeczywistych danych, które zostały zebrane za pomocą potoku przedstawionego w rozdziale 3. Duży model językowy został użyty do konwersji nieprzetworzonego tekstu na postać par instrukcji, które składają się z pytania i udzielonej na nie odpowiedzi. Następnie omówiliśmy techniki nadzorowanego dostrajania modelu. Przedstawiliśmy między innymi analizę wad i zalet nadzorowanego dostrajania modelu, metod przechowywania i przetwarzania zbiorów danych instrukcji za pomocą szablonu czatu oraz przegląd trzech podstawowych technik nadzorowanego dostrajania modelu: pełne dostrajanie, LoRA i QLoRA. Metody te porównaliśmy pod względem wpływu na poziom użycia pamięci, efektywność trenowania i jakość danych wyjściowych. Na końcu rozdziału przedstawiliśmy przykład praktyczny, który pokazał dostrojenie modelu Llama 3.1 8B za pomocą samodzielnie przygotowanego zbioru danych instrukcji. W tym przykładzie podkreśliliśmy ważne kroki i szczegóły implementacji, dzięki którym proces dostrajania ma szansę zakończyć się sukcesem.

W następnym rozdziale użyjemy techniki uwzględniania preferencji, aby utworzyć nową wersję modelu Twin-Llama-3.1-8B. Wygenerujemy nowy zbiór danych z wybranymi i odrzuconymi odpowiedziami, co pomoże skalibrować typy odpowiedzi oczekiwane od modelu. Dokładnie wyjaśnimy wiele zastosowań, które mogą czerpać korzyści z takiego frameworka. Pokażemy również, jak go zaimplementować.

Źródła

- Tahori, Gulrajani, Zhang, Dubois i inni, *Alpaca: A Strong, Replicable Instruction-Following Model*, crfm.stanford.edu, <https://crfm.stanford.edu/2023/03/13/alpaca.html>, 13 marca 2023.
- Mukherjee Subhabrata i inni, *Orca: Progressive Learning from Complex Explanation Traces of GPT-4*, arXiv preprint arXiv:2306.02707, czerwiec 2023.
- Lian Wing, Goodson Bleys, Pentland Eugene, Cook Austin, Vong Chanvichet, „Teknium”, *Open-Orca/OpenOrca*, <https://huggingface.co/datasets/Open-Orca/OpenOrca>, 2023.

- Zeng Weihao i inni, *Automatic Instruction Evolving for Large Language Models*, arXiv preprint arXiv:2406.00770, czerwiec 2024.
- Zhou Chunting i inni, *LIMA: Less Is More for Alignment*, arXiv preprint arXiv:2305.11206, maj 2023.
- 01. AI, *Yi: Open Foundation Models by 01.AI*, arXiv preprint arXiv:2403.04652, marzec 2024.
- Birch Alex, *LLM finetuning memory requirements*, <https://blog.scottlogic.com/2023/11/24/llm-mem.html>, 24 listopada 2023.
- Anthony Quentin i inni, *Transformer Math 101*, <https://blog.eleuther.ai/transformer-math/>, 18 kwietnia 2023.
- Hu Edward J. i inni, *LoRA: Low-Rank Adaptation of Large Language Models*, arXiv preprint arXiv:2106.09685, czerwiec 2021.
- Dettmers Tim i inni, *QLoRA: Efficient Finetuning of Quantized LLMs*, arXiv preprint arXiv:2305.14314, maj 2023.

PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

Działa? To za mało. Musi działać dobrze!

Wraz ze wzrostem popularności dużych modeli językowych rośnie zapotrzebowanie na specjalistów, którzy potrafią je skutecznie wdrażać w rzeczywistych rozwiązaniach. Inżynieria LLM to szeroki zestaw zadań, wymagający unikalnego połączenia wiedzy z wielu dziedzin. Szczególnie istotne okazuje się tu podejście MLOps, które znacząco zwiększa szanse na sukces w projektach opartych na modelach językowych.

Ten obszerny przewodnik pokaże Ci, jak stosować najlepsze praktyki w pracy z LLM. Znajdziesz tu omówienie kluczowych koncepcji, praktyczne techniki i porady ekspertów z zakresu inżynierii danych, dostrajania i oceny modeli, optymalizacji wnioskowania, a także budowania skalowalnych potoków przetwarzania. Krok po kroku prześledzisz, jak zrealizować konkretny produkt, integrując różne aspekty inżynierii LLM i metodologię MLOps. Dowiesz się, jak zbierać i przygotowywać dane, dostrajać modele do specyficznych zastosowań, zwiększać ich wydajność i wdrażać rozwiązania oparte na technice RAG.

Najciekawsze zagadnienia:

- niezawodne potoki danych i zarządzanie cyklem trenowania modeli LLM
- tworzenie i udoskonalanie modeli językowych w praktyce
- podejście MLOps — koordynacja komponentów i monitorowanie promptów
- nadzorowane dostrajanie i ewaluacja modeli
- wykorzystanie narzędzi chmurowych (na przykład AWS) w skalowalnych wdrożeniach
- praktyczne zastosowanie techniki retrieval-augmented generation (RAG)

Paul Iusztin pełni funkcję starszego inżyniera uczenia maszynowego w firmie Metaphysic. Jest założycielem kanału edukacyjnego Decoding ML, poświęconego tworzeniu systemów uczenia maszynowego.

Dr Maxime Labonne kieruje zespołem Post-Training w Liquid AI. Jest uznanym ekspertem Google w dziedzinie sztucznej inteligencji. Organizuje kursy dotyczące dużych modeli językowych i pracuje nad modelami takimi jak NeuralDaredevil.

**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 978-83-289-2530-4



9 788328 925304

Cena: 119,00 zł

<packt>