O'REILLY®

LLM w środowisku produkcyjnym

Serwowanie, optymalizacja i hosting na dużą skalę



Chi Wang
Peiheng Hu

Helion

Tytuł oryginału: Hands-On LLM Serving and Optimization: Hosting LLMs at Scale

Tłumaczenie: Robert Górczyński

ISBN: 978-83-289-4204-2

© 2027 Helion S.A.

Authorized Polish translation of the English edition of Hands-On LLM Serving and Optimization ISBN 9798341621497© 2026 Chi Wang and Peiheng Hu.

This translation is published and sold by permission of O'Reilly Media, Inc.,
which owns or controls all rights to publish and sell the same.

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/llmsro
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

Wprowadzenie 9

1. Wprowadzenie do serwowania i optymalizacji modeli 19
Anatomia modelu 20
Architektura modelu 22
Dane modelu 22
Kod wykonywalny modelu 22
Cykl życiowy modelu — od trenowania do serwowania 23
Czym jest serwowanie modeli? 24
Dlaczego warto uczyć się serwowania modeli? 26
Dlaczego należy optymalizować serwowanie modeli (szczególnie dużych modeli językowych)? 29
Przykład wykorzystania frameworka do serwowania modeli (vLLM) w celu poprawy przepustowości dużego modelu językowego 30
Paradygmaty serwowania modeli 33
Serwowanie w urządzeniach (Edge) 33
Usługa pojedynczego modelu 38
Usługa wielomodelowa 43
Platformy do serwowania modeli 47
Podsumowanie 48

2. Serwowanie dużych modeli językowych 51
Wewnątrz transformera 52
Ewolucja dużego modelu językowego 52
Autoregresywna natura transformerów 55
Architektura transformera oparta wyłącznie na dekoderze 57
Uchwycenie kontekstu tokena poprzez obliczenie uwagi 62
Generowanie dużego modelu językowego — przewodnik krok po kroku 65
Uruchamianie modelu Qwen 66
Predykcja wiersz po wierszu 67
Włącz pamięć podręczną typu KV, aby zwiększyć wydajność działania 69

Kup książkę Poleć książkę

Fazy wstępnego wypełniania i dekodowania 73
Uruchamianie dużego modelu językowego za pomocą frameworka serwującego 75
Serwowanie dużego modelu językowego (Qwen) za pomocą vLLM 76
Porównanie wydajności działania — vLLM kontra Hugging Face Transformers 77
Podstawy serwowania strumieniowego LLM-a 78
Podstawy przetwarzania wsadowego podczas serwowania dużych modeli językowych 80
Podsumowanie 82

3. Projektowanie systemu serwowania modeli — dokładna analiza 83
Tworzenie od podstaw działającej online usługi serwowania dużego modelu językowego 84
Cele projektowe 84
Architektura usługi 85
Implementacja obsługi pojedynczego żądania generowania 87
Przetwarzanie wsadowe 90
Strumieniowanie z przetwarzaniem wsadowym 95
Serwowanie wsadowe za pomocą vLLM 100
Ogólny projekt serwowania pojedynczego dużego modelu językowego 102
Wymagania związane z serwowaniem pojedynczego modelu 103
Projekt ogólny 104
Tworzenie od podstaw usługi serwowania wielu modeli 107
Cele projektowe 108
Architektura usług 108
Podstawowa implementacja 110
NVIDIA Triton jako serwer modeli 113
Kompromisy w projektach serwowania wielu modeli 117
Wyzwania 117
Projekt wielomodelowy zoptymalizowany pod kątem kosztów 118
Projekt wielomodelowy zoptymalizowany pod kątem opóźnień 119
Podsumowanie 121

4. Najlepsze praktyki w zakresie serwowania modeli 123
Serwowanie modeli w świecie agentów 124
Definicja agenta 125
Przykładowy agent wiedzy 126
Projekt agenta 127
Wewnętrzny przepływ pracy agenta 128
Autonomia agenta 130
Generowanie za pomocą systemu RAG 132
Generowanie wspomagane pamięcią podręczną (CAG) 134
Jak agenty korzystają z serwowania modeli? 137

Kup książkę Poleć książkę

Serwowanie dużych modeli językowych w systemach korporacyjnych — ogólny przegląd 137
Warstwa publicznego API 139
Warstwa zarządzania zasobami 139
Warstwa wyboru modelu i koordynacji 139
Rozproszona warstwa serwująca 140
Główna warstwa wnioskowania 141
Warstwa optymalizacji modelu 141
Warstwa modelu 141
Tworzenie rozwiązania za pomocą stosu otwartoźródłowego 141
Implementacja publicznego API 143
Implementacja wyboru modelu 144
Implementacja punktu końcowego serwującego model 145
Budowanie rozwiązania z użyciem dostawcy chmury 149
Opcja 1. W pełni zarządzane serwowanie modeli bazowych 150
Opcja 2. Wdrożenie modelu podstawowego jednym kliknięciem 151
Opcja 3. Skorzystanie z własnego modelu 153
Opcja 4. Skorzystanie z własnego kodu 156
Opcja 5. Skorzystanie z własnego obrazu serwowania 159
Opcja 6. Przygotowanie własnej infrastruktury serwowania 160
Porównanie opcji 161
Budować czy kupować? Zrozumienie strategii 161
Dlaczego wiedza o tym, jak budować, pomaga (nawet jeśli nie będziesz budować)? 163
Nasza strategia wyboru 163
Mierzenie wydajności serwowania dużych modeli językowych 164
Wskaźniki opóźnienia 165
Wskaźniki przepustowości 167
Najlepsze praktyki pomiaru wydajności działania 168
Podsumowanie 170

5. Wyzwania związane z serwowaniem dużych modeli językowych 173
Dlaczego optymalizacja serwowania dużych modeli językowych jest ważna? 174
Wrażenia klienta 175
Efektywność kosztowa 176
Skalowalność, obsługa obciążenia szczytowego i wykonalność 178
Rola akceleratorów w serwowaniu dużego modelu językowego 179
Czytanie specyfikacji GPU 179
Porównanie specyfikacji popularnych układów graficznych 186
Wąskie gardła podczas wczytywania dużego modelu językowego 188
Proces wczytywania modelu 188
Szacowanie wielkości modelu 189
Szacowanie wielkości pamięci podręcznej typu KV 190

Kup książkę Poleć książkę

Wąskie gardła podczas wykonywania dużego modelu językowego 194
Granice mocy obliczeniowej układu graficznego i przepustowości jego pamięci 194
Intensywność arytmetyczna w mnożeniu macierzy 197
Zastosowanie analizy intensywności arytmetycznej dla faz wstępnego wypełniania i dekodowania w dużym modelu językowym 199
Inne akceleratory AI i trendy 202
Podsumowanie 205

6. Podstawowe techniki optymalizacji dużych modeli językowych 207
Optymalizacja poprzez grupowanie żądań i tworzenie harmonogramów 207
Dlaczego potrzebujemy przetwarzania wsadowego podczas serwowania w czasie rzeczywistym? 208
Dynamiczne tworzenie partii danych podczas wnioskowania w trybie online 209
Ciągłe przetwarzanie wsadowe dla wnioskowania w trybie online z użyciem dużego modelu językowego 210
Ciągłe przetwarzanie wsadowe z podzieloną na fragmenty fazą wstępnego wypełniania 214
Skalowanie mechanizmu uwagi i optymalizacja jądra 217
Skalowalne mechanizmy uwagi 218
Fuzja jądra i niestandardowe jądra uwagi 220
Kompresja modelu 224
Kwantyzacja 225
Destylacja 240
Przycinanie 241
Buforowanie prefiksów 242
RadixAttention 244
Przypadki użycia 244
Najlepsze praktyki 246
Skalowanie buforowania prefiksów 247
Podsumowanie 249

7. Zaawansowane techniki optymalizacji dużych modeli językowych 251
Dekodowanie spekulatywne 251
Szczegółowe kroki 252
Dostrojenie i użycie 253
Praktyczne zastosowanie dekodowania spekulatywnego 258
Wnioskowanie z użyciem wielu węzłów i procesorów graficznych 260
Równoległość danych 261
Równoległość tensorów i potoków 263
Równoległość ekspertów 268

Kup książkę Poleć książkę

Rozdzielona obsługa faz wstępnego wypełniania i dekodowania 269
Ogólna architektura 271
Przekazanie pamięci podręcznej typu KV 272
Kiedy używać rozdzielonej obsługi faz wstępnego wypełniania i dekodowania? 275
Zaawansowane buforowanie typu KV 275
Obsługa długiego kontekstu 277
Obliczenia kosztów i opóźnień 278
Samodzielny hosting dużych modeli językowych 280
Praktyczne zastosowanie LMCache 283
Podsumowanie 287

8. Frameworki do serwowania dużych modeli językowych 289
Dlaczego potrzebujemy wyspecjalizowanych frameworków do serwowania dużych modeli językowych? 290
vLLM 291
Architektura vLLM 292
Przepływ pracy inicjalizacji modelu (z wieloma procesami roboczymi) 294
Przepływ pracy dla żądania generowania 296
Szczegółowa analiza komponentu Scheduler 298
Warstwowa strategia optymalizacji w vLLM 302
TensorRT-LLM 303
SGLang 304
llama.cpp 305
Wybór odpowiedniego frameworka 307
Podsumowanie 309

9. Optymalizacja dużych modeli językowych w praktyce 311
Plan optymalizacji serwowania dużego modelu językowego 312
Optymalizacja serwowania Qwen3-14B za pomocą vLLM 314
Krok 1. Sprawdzenie procesora graficznego 314
Krok 2. Generowanie ruchu testowego 315
Krok 3. Zdefiniowanie wskaźników oceny 318
Krok 4. Konfiguracja serwera odpowiadającego za serwowanie modelu 319
Krok 5. Testowanie wydajności modelu Qwen3 za pomocą vLLM 320
Krok 6. Testowanie wydajności kwantyzowanego modelu Qwen3 serwowanego za pomocą vLLM 322
Krok 7. Zastosowanie dodatkowych technik optymalizacji 324
Krok 8. Testowanie wydajności modelu Qwen3 z wykorzystaniem serwowania rozproszonego 325
Typowe kompromisy optymalizacyjne 329
Podsumowanie 330

Kup książkę Poleć książkę

10. Postępy w serwowaniu dużych modeli językowych 333
Buforowanie semantyczne 334
Strategie profilowania wydajności 336
Serwowanie wielomodalne 339
Przetwarzanie wielomodalnych danych wejściowych 340
Implikacje dla architektury i systemu 341
Sztuczna inteligencja na brzegu sieci — czynniki napędzające i umożliwiające 342
Wyspecjalizowany sprzęt o niskim poborze mocy 343
Kompresja i optymalizacja modeli 344
Obliczenia heterogeniczne 344
Harmonogramowanie uwzględniające temperaturę 345
Przetwarzanie hybrydowe w urządzeniach brzegowych i w chmurze 345
Serwowanie wielu modeli LoRA 346
Serwowanie modeli w uczeniu przez wzmacnianie 348
Serwowanie dużych modeli językowych w uczeniu przez wzmacnianie 348
Determinizm w serwowaniu RL 349
Podsumowanie 350

Kup książkę Poleć książkę

ROZDZIAŁ 1.

Wprowadzenie do serwowania i optymalizacji modeli

W ciągu ostatniej dekady systemy sztucznej inteligencji ewoluowały z prototypów badawczych, działających w trybie offline, do funkcji czasu rzeczywistego wbudowanych w produkty, z których codziennie korzystają użytkownicy. Nowoczesne przepływy pracy sztucznej inteligencji obejmują pełny cykl życia — od gromadzenia danych i trenowania modeli po wdrażanie, monitorowanie i ciągłą iterację — a cykl ten znacznie przyspieszył wraz z rozwojem uczenia głębokiego i dużych modeli językowych (LLM). Chociaż trenowanie coraz potężniejszych modeli przyciąga wiele uwagi, niezawodne i wydajne dostarczanie tych modeli do środowiska produkcyjnego stało się równie krytyczne.

W swojej istocie *serwowanie modeli* (ang. *model serving*) to proces, który rozwiązuje problem serwowania modeli sztucznej inteligencji użytkownikom końcowym, aplikacjom i systemom. Ten proces działa za pośrednictwem API, usług internetowych lub zintegrowanych przepływów pracy w celu generowania predykcji (zwanych *wnioskowaniem*) na nowych i niewidzianych wcześniej danych.

Używając prostej analogii: dla firm wszelkiego rodzaju — niezależnie od tego, czy ich celem jest dostarczanie możliwości sztucznej inteligencji klientom, czy zwiększanie wydajności operacyjnej — serwowanie modeli jest formą łańcucha dostaw. Wytrenowany model ma niewielką wartość biznesową, jeżeli nie może zostać dostarczony użytkownikom z akceptowalnym opóźnieniem, niezawodnością i kosztem. Na przykład Amazon i Netflix używają serwowania modeli do natychmiastowego aktualizowania rekomendacji dla klientów podczas przeglądania treści. Banki używają serwowania modeli do blokowania nieuczciwych transakcji podczas płatności w sklepach internetowych, a chatboty linii lotniczych używają go do dostarczania natychmiastowych aktualizacji lotów i opcji zmiany rezerwacji. Działalność tych firm zostaje wstrzymana, jeśli ich systemy serwowania modeli ulegną awarii.

Podobnie jak każdy producent dąży do przygotowania wydajnego i opłacalnego łańcucha dostaw, firmy zajmujące się sztuczną inteligencją starają się wydajnie i skutecznie wykorzystywać swoje modele i sprzęt w środowiskach produkcyjnych. Dlatego wybór odpowiedniego podejścia do serwowania modeli i jego optymalizacja są kluczowe: ma to bezpośredni wpływ

Kup książkę Poleć książkę

na linię życia firmy i koszty operacyjne. Niezależnie od pełnionej roli, jeśli pracujesz w branży sztucznej inteligencji, odniesiesz korzyści z posiadania wiedzy na temat serwowania modeli.

Jako liderzy technologiczni, którzy od ponad dekady pracują nad infrastrukturą serwowania modeli, współpracowaliśmy z różnymi uczestnikami w tej dziedzinie: badaczami, programistami, kadrą kierowniczą, marketingowcami, klientami i studentami. Odkryliśmy, że ludzie często czują się onieśmieleni lub przytłoczeni, gdy po raz pierwszy próbują zrozumieć serwowanie modeli. Istnieją trzy główne powody takiego stanu rzeczy. Po pierwsze powinieneś już mieć rozległą wiedzę na temat trenowania modeli. Po drugie jak dotąd nie ma jasnej ścieżki uczenia się, która prowadzi od samouczków dla początkujących do zarządzania światowej klasy systemem udostępniania. Po trzecie istnieje tak wiele frameworków, bibliotek, dostawców i innych opcji inżynieryjnych, że trudno jest podjąć decyzję, z których z nich skorzystać. Ta książka ma na celu sprostanie powyższym wyzwaniom oraz zapewnić ustrukturyzowany, praktyczny przewodnik po udostępnianiu i optymalizowaniu modeli — taki, który wypełnia lukę między teorią i praktyką, a jednocześnie umożliwia czytelnikom podejmowanie świadomych decyzji.

W tym rozdziale kładziemy podwaliny pod zrozumienie pozostałej części książki. Rozpoczynamy od wyjaśnienia podstawowych pojęć związanych z udostępnianiem modeli, a następnie omawiamy, dlaczego solidne, zoptymalizowane serwowanie modeli jest bardzo ważne w rzeczywistych zastosowaniach. Na koniec przedstawimy ogólne paradygmaty w zakresie serwowania modeli. Po lekturze tego rozdziału zyskasz kompleksowy przegląd zagadnień związanych z udostępnianiem i optymalizowaniem modeli, co przygotowuje grunt pod materiały praktyczne w kolejnych rozdziałach.

Anatomia modelu

Z akademickiego punktu widzenia *model uczenia maszynowego* to matematyczna reprezentacja lub algorytm, który uczy się wzorców z danych w celu dokonywania predykcji, podejmowania decyzji lub wyciągania wniosków bez jawnego programowania do tego zadania.

W inżynierii i operacjach skoncentrujemy się bardziej na tym, jak używać modeli, niż na tym, jak je trenować. Dlatego przez większość czasu traktujemy modele jako czarne skrzynki — zbiory plików (wykonywalnych) wytworzonych w procesach uczenia maszynowego.

Modele postrzegamy jako składające się z trzech typów plików, mianowicie danych, architektury i kodu wykonywalnego (rysunek 1.1):

Dane modelu

Dane modelu obejmują jego wagi, obciążenia i konfigurację. *Wag i obciążeń* model uczy się podczas treningu, natomiast *konfiguracja* modelu zawiera metadane przeznaczone do uruchomienia modelu, takie jak jego osadzenia i klasy etykiet (dla modeli klasyfikacji), jego właściwość `max_batch_size` (dla wnioskowania wsadowego) oraz jego tensory wejściowe i wyjściowe.

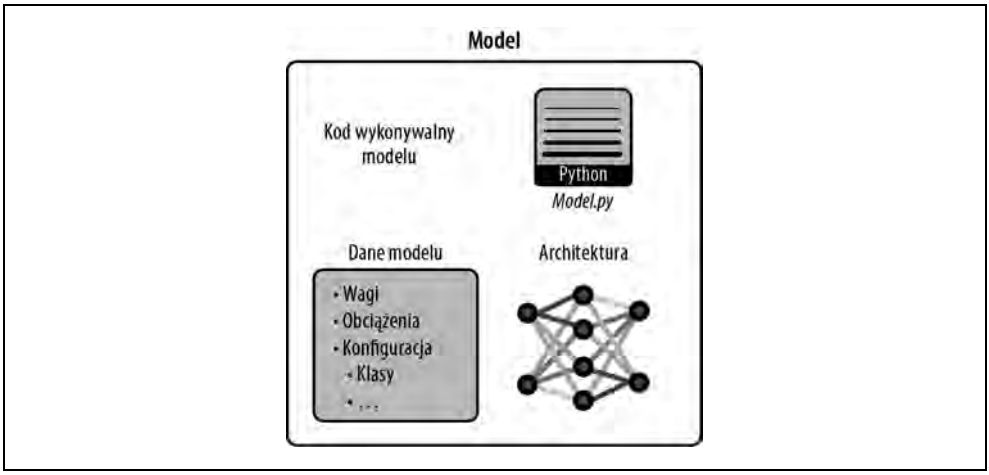
Kup książkę Poleć książkę

Architektura modelu

Architektura odnosi się do struktury i projektu modelu uczenia maszynowego. Definiuje, jak model jest zorganizowany, w tym typy i liczbę warstw, połączenia między warstwami oraz operacje, które model wykonuje. Architektura określa, w jaki sposób model przetwarza dane wejściowe w celu wygenerowania predykcji wyjściowych lub decyzji.

Kod wykonywalny modelu

Kod wykonywalny modelu jest tym, co model uruchamia. Zazwyczaj inicjuje architekturę we frameworku serwującym model, wczytuje wagi i dokonuje predykcji (lub generuje inne dane wyjściowe).



Rysunek 1.1. Model składa się z architektury, kodu wykonywalnego i danych



Modele jako programy wykonywalne, a nie dane statyczne

Modele można traktować jako pliki wykonywalne programów. Wiele osób błędnie zakłada, że modele to jedynie pasywne pliki z danymi, podczas gdy w rzeczywistości zawierają one również logikę wykonania, metadane i inne komponenty, które definiują sposób ich działania. Zamiast być statycznymi artefaktami, modele funkcjonują jako programy dynamiczne, które mogą przetwarzać dane wejściowe, podejmować decyzje i ewoluować w czasie.

Przeanalizujmy strukturę modelu z rysunku 1.1 na przykładzie z samouczka PyTorch (*https://pytorch.org/*) „Saving and Loading Models” (*https://github.com/pytorch/tutorials/blob/main/beginner_source/saving_loading_models.py*), który wytrenowany model zapisuje na dysku lokalnym.

Kup książkę Poleć książkę

Architektura modelu

Po pierwsze zamieszczony tutaj plik w języku Python definiuje architekturę modelu. Ten kod (znajdujący się w pliku/pakiecie modelu) jest zazwyczaj kopią klasy architektury modelu w kodzie do szkolenia modelu:

```
class TheModelClass(nn.Module):
    def __init__(self):
        super(TheModelClass, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Dane modelu

Po drugie po wytrenowaniu modelu wagi i obciążenia są zapisywane w pliku *model_weights.pt*:

```
// W PyTorch parametry uczenia się (np. wagi i obciążenia)
// są zapisywane w pliku „słownika stanu”
torch.save(model.state_dict(), model_weights.pt)
```

Kod wykonywalny modelu

Po trzecie przedstawiony tutaj przykładowy fragment kodu definiuje sposób wykonania modelu dla podanych danych wejściowych:

```
model = TheModelClass(*args, **kwargs) // Inicjalizacja modelu
model.load_state_dict(torch.load("model_weights.pt", weights_only=True))
        // Wczytanie wag modelu ("model_weights.pt")
model.eval() // Przypisanie modelowi trybu oceny w celu wykonania predykcji
pred = model(inputs) // Rozpoczęcie przez model tworzenia predykcji na podstawie dostarczonych danych
wejściowych
```

Chociaż poszczególne frameworki uczenia maszynowego, takie jak TensorFlow i PyTorch, udostępniają różne API i biblioteki, podstawowe zasady pakowania i struktury modelu (rysunek 1.1) są w dużej mierze podobne. W praktyce często stosuje się dodatkowe optymalizacje plików modeli w celu zwiększenia wydajności serwowania, szczególnie w przypadku dużych modeli językowych. Optymalizacje te omawiamy w rozdziałach 6. i 7.

Kup książkę Poleć książkę



Zalecamy przechowywanie architektury i danych modelu w oddzielnych plikach

Chociaż możliwe jest zapisanie całego modelu w jednym pliku (na przykład `torch.save(model, "model.pt")`), to jednak większą elastyczność zapewnia oddzielenie definicji klasy (architektury) od wag.

Taka separacja umożliwia bardziej zaawansowane scenariusze serwowania. Na przykład gdy architektura modelu nieznacznie zmienia się między wersjami, niektóre parametry w punkcie kontrolnym modelu mogą już nie odpowiadać dokładnie nowej strukturze modelu. Są one określane jako *klucze niedopasowane*. W takich przypadkach frameworki typu PyTorch umożliwiają tak zwane *wczytywanie częściowe*, w którym parametry zgodne są wczytywane, natomiast niekompatybilne są pomijane. Jest to powszechnie stosowane podczas dostrajania, rozszerzania modelu o dodatkowe warstwy lub wprowadzania przyrostowych aktualizacji architektury bez konieczności ponownego trenowania modelu od zera.

Skoro rozumiesz, czym jest model uczenia maszynowego, przyjrzyjmy się ogólnemu procesowi trenowania i serwowania modelu.

Cykl życiowy modelu — od trenowania do serwowania

Zanim zagłębimy się w serwowanie modeli, warto spojrzeć z szerszej perspektywy i zrozumieć, gdzie serwowanie wpisuje się w szerszy cykl życiowy uczenia maszynowego. Większość dyskusji na temat sztucznej inteligencji koncentruje się w dużej mierze na trenowaniu modeli: zbieraniu danych, projektowaniu architektury i poprawianiu dokładności. Jednak w rzeczywistych systemach trenowanie to tylko jeden z etapów. Prawdziwą wartość modelu uczenia maszynowego można uświadomić sobie dopiero wtedy, gdy zostanie on wdrożony, a następnie będzie niezawodnie serwowany i wydajnie obsługiwany na dużą skalę.

Na wysokim poziomie cykl życiowy uczenia maszynowego składa się zazwyczaj z następujących etapów:

Zbieranie danych

Nieprzetworzone dane są gromadzone ze źródeł takich jak interakcje użytkowników, czujniki, dzienniki zdarzeń, dokumenty lub od dostawców zewnętrznych. Te dane są następnie czyszczone, etykietowane i selekcjonowane w celu utworzenia zbiorów danych treningowych i testowych.

Trenowanie i dostrajanie

Modele są trenowane w celu uczenia się wzorców z danych. W przypadku nowoczesnych systemów często obejmuje to wstępne trenowanie dużych modeli bazowych i ich dostrajanie na podstawie danych specyficznych dla konkretnej dziedziny, aby spełnić wymagania konkretnej aplikacji.

Ocena

Wytrenowane modele są sprawdzane pod kątem wskaźników offline, takich jak dokładność, strata lub testy wydajności ściśle związane z konkretnym zadaniem. Ten etap pomaga określić, czy model jest gotowy do użycia w środowisku produkcyjnym.

Kup książkę Poleć książkę

Wdrożenie

Wytrenowany model jest pakowany i udostępniany w środowisku produkcyjnym. W tym momencie model staje się artefaktem oprogramowania, który musi być wersjonowany, testowany i zintegrowany z istniejącymi systemami.

Serwowanie (temat tej książki)

Wdrożony model jest udostępniany aplikacjom, użytkownikom lub innym systemom — zazwyczaj za pośrednictwem API lub usług — i dzięki temu może, w czasie rzeczywistym lub na dużą skalę, generować predykcje (wnioskowanie) na nowych, niewidzianych wcześniej danych.

Optymalizacja i iteracja

Gdy model obsługuje ruch na żywo, zespoły stale optymalizują wydajność serwowania modelu, jego niezawodność i koszty. Po stronie trenowania informacje zwrotne z użytkowania w środowisku produkcyjnym często napędzają dalsze dostrajanie, ponowne trenowanie lub zmiany architektoniczne, co powoduje ponowne rozpoczęcie cyklu życiowego.

Uczenie modelu to nie to samo co uruchomienie go w środowisku produkcyjnym. Ograniczenia, cele i wyzwania inżynieryjne zmieniają się radykalnie, gdy model zaczyna obsługiwać prawdziwych użytkowników.

Czym jest serwowanie modeli?

W terminologii inżynierskiej *serwowanie modelu* oznacza wdrożenie modelu uczenia maszynowego w środowisku produkcyjnym, w którym może on przetwarzać nowe dane i generować predykcje[1]. Wiąże się to z konfiguracją niezbędnej infrastruktury — zarówno oprogramowania, jak i sprzętu — aby model mógł odbierać dane wejściowe, przeprowadzać wnioskowanie oraz zwracać wyniki w sposób wydajny, skalowalny i niezawodny.

Serwowanie modelu może odbywać się *lokalnie* (w urządzeniu), w *klastrze* (we własnej infrastrukturze) lub *zdalnie* (w chmurze), w zależności od wymagań biznesowych i ograniczeń infrastruktury. Oto trzy przykłady, po jednym dla każdego scenariusza serwowania:

Robot magazynowy (w urządzeniu)

Model wykrywania obiektów w czasie rzeczywistym, taki jak YOLO (*https://arxiv.org/abs/1506.02640*), jest wdrażany w komputerze pokładowym robota (powiedzmy Raspberry Pi z Coral TPU). To pozwala robotowi przetwarzać dane wizualne i natychmiastowo podejmować autonomiczne decyzje.

Wyszukiwanie dokumentów (we własnej infrastrukturze)

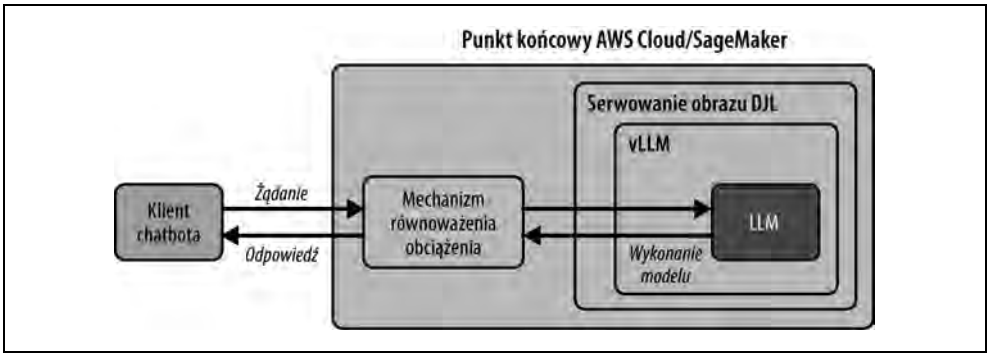
Aby zbudować semantyczny indeks wyszukiwania dla wewnętrznych danych firmy, model językowy (taki jak Sentence-BERT) jest serwowany w klastrze obliczeniowym firmy. Taka konfiguracja generuje gęste osadzenia wektorowe dla dokumentów, a jednocześnie zapewnia bezpieczeństwo danych i zgodność z przepisami poprzez unikanie transferów danych na dużą skalę.

[1] W tej książce zamiennie używamy terminów *serwowanie modelu*, *wnioskowanie modelu* i *predykcja*.

Kup książkę Poleć książkę

Chatbot obsługi klienta (w chmurze)

Gdy klient przesyła zapytanie, chatbot wysyła je wraz z odpowiednimi dokumentami do zdalnego dużego modelu językowego hostingowanego lub dostarczanego przez dostawcę, takiego jak Amazon SageMaker Inference (*https://docs.aws.amazon.com/sagemaker/latest/dg/deploy-model.html*) bądź OpenAI (*https://openai.com/*), jak pokazaliśmy na rysunku 1.2.



Rysunek 1.2. Serwowanie dużego modelu językowego dla aplikacji chatbota w punkcie końcowym platformy SageMaker

W kontekście serwowania modeli inżynierowie ML i DevOps zazwyczaj w *mniejszym* stopniu koncentrują się na wewnętrznych szczegółach modeli — takich jak ich architektura, metody uczenia i formaty plików — i traktują modele jak czarne skrzynki. To wynika z faktu, że nowoczesne środowiska przeznaczone do serwowania modeli, takie jak vLLM i TensorRT-LLM, radzą sobie ze złożonością wykonywania modeli, udostępniając warstwę abstrakcji, dzięki której wykonywanie predykcji przez model jest prostsze i bardziej wydajne.

W przypadku serwowania modeli koncentrujemy się bardziej na następujących kwestiach:

Wdrażanie

Wybór odpowiedniego sprzętu i udostępnienie modelu z potoku uczenia (lub z opcji otwartoźródłowych) w celu pobierania danych wejściowych i zwracania predykcji.

Skalowalność i dostępność

Zdolność do wydajnej obsługi od kilku tysięcy do milionów żądań przy jednoczesnym zapewnieniu spójnych wrażeń klienta.

Opóźnienie

Szybkie dostarczanie predykcji — w milisekundach, w przypadku przypadków użycia w czasie rzeczywistym.

Monitorowanie

Śledzenie wydajności działania modelu, dryfu danych i kondycji systemu.

Wersjonowanie

Zarządzanie aktualizacjami i wycofywaniem modeli bez zakłócania pracy klientów.

Bezpieczeństwo

Dane wrażliwe muszą być chronione, a dostęp do modelu powinien być kontrolowany.

Kup książkę Poleć książkę

Koszt obsługi

Najbardziej decydujący czynnik ze wszystkich. Kosztu obsługi używamy jako kluczowego czynnika do oceny różnych podejść w zakresie serwowania modelu i związanych z tym kompromisów.

W praktyce inżynierowie zajmujący się obsługą traktują większość modeli uczenia głębokiego (wyjątkiem są LLM-y) jako wdrożone artefakty oprogramowania. Koncentrują się na tym, jak pakować, wdrażać, skalować, monitorować i optymalizować obciążenia związane z wnioskowaniem, aby modele konsekwentnie spełniały biznesowe umowy o gwarantowanym poziomie świadczenia usług (ang. *service-level agreement*, SLA) i jednocześnie dostarczały wartość aplikacjom i użytkownikom końcowym. Ta praca w mniejszym stopniu opiera się na wymyślaniu nowych algorytmów uczenia się, natomiast w większym na inżynierii systemów, projektowaniu infrastruktury i kompleksowej optymalizacji wydajności działania w stosach sprzętowych i programowych.



Wiedza o dużych modelach językowych ma kluczowe znaczenie dla ich serwowania

Do efektywnego serwowania i optymalizowania dużych modeli językowych niezbędne jest solidne zrozumienie algorytmów sztucznej inteligencji, w szczególności transformerów (*https://arxiv.org/abs/1706.03762*). Wszystkie postępy w technikach serwowania LLM-ów mają na celu przezwyciężenie wąskich gardeł w ich wykonywaniu. Dlatego dobre zrozumienie architektury LLM-ów dostarcza informacje potrzebne do optymalizacji przepustowości i opóźnień. W kolejnych rozdziałach omówimy podstawową wiedzę, która pomoże zrozumieć metody serwowania i optymalizacji LLM-ów.

Dlaczego warto uczyć się serwowania modeli?

Wprawdzie większość osób zgodzi się, że serwowanie modeli jest ważne, ale nie mają pewności, dlaczego muszą uczyć się budować lub dostosowywać własne systemy serwowania modeli. W tym podrozdziale odpowiemy na niektóre z tych pytań i będziemy opierać się na własnych doświadczeniach.

Pytanie 1: „Korzystamy z modeli pochodzących bezpośrednio od dostawców chmury, takich jak Amazon AWS (SageMaker lub Bedrock) i Microsoft Azure ML. Dlaczego mielibyśmy zajmować się budowaniem systemu serwowania modeli?”.

Wdrażanie i hosting modeli na platformach dostawców chmury to w wielu sytuacjach dobre rozwiązanie. Usługa chmury może zająć się wieloma kwestiami, takimi jak zarządzanie sprzętem i instalowanie poprawek oprogramowania, a także umożliwia na przykład szybkie potwierdzanie koncepcji. Z powodzeniem zaprojektowaliśmy i zbudowaliśmy wiele rozwiązań przeznaczonych do serwowania modeli na platformach AWS i Azure. Jednak nigdy nie można po prostu korzystać z tych usług bez starannego przemyślenia.

Chociaż oparte na chmurze rozwiązania w zakresie serwowania modeli są wygodne, wiążą się również z wyzwaniami. Po pierwsze istnieje wiele produktów i funkcji do wyboru, co utrudnia

Kup książkę Poleć książkę

określenie najlepszego dopasowania. Po drugie integracja usług dostawców z istniejącą infrastrukturą i aplikacjami klienckimi często wymaga dodatkowych nakładów inżynieryjnych. Po trzecie często trzeba optymalizować i dostrajać usługi dostawców, aby dostosować je do konkretnych przypadków użycia biznesowego i wymagań dotyczących wydajności.

Dostawcy chmury projektują swoje rozwiązania do serwowania modeli w oparciu o różne potrzeby klientów i udostępniają je w postaci produktów ogólnego przeznaczenia. Na przykład usługa AWS SageMaker zapewnia wiele opcji serwowania, w tym punkty końcowe dla jednego i wielu modeli, poszczególne opcje zaś wiążą się z różnymi poziomami dostosowywania, kompromisami i kosztami. Bez jasnego zrozumienia sposobu serwowania modeli, a także związanych z tym wyzwań i przypadków użycia bardzo łatwo jest dokonać nieoptymalnych wyborów, które mogą zwiększyć koszty lub ograniczyć elastyczność.

Krótko mówiąc, zrozumienie, jak jest budowany i obsługiwany system serwowania modeli, pomaga w najlepszym wykorzystaniu opcji serwowania modeli oferowanych przez dostawców chmury.

Pytanie 2: „Podstawowe LLM-y są na tyle potężne, że nie musimy korzystać z żadnych innych. Po prostu pozwalamy naszej aplikacji klienckiej wywoływać dostawców modeli podstawowych, takich jak OpenAI, DeepSeek i Anthropic. W jakim celu mielibyśmy budować i utrzymywać stos serwowania modeli?”.

LLM-y to jeden z najważniejszych wynalazków w dziedzinie sztucznej inteligencji i kamień milowy na drodze do ogólnej sztucznej inteligencji (AGI). Jednak ich zastosowanie w biznesie wiąże się z wieloma niuansami.

Przede wszystkim pod uwagę należy wziąć koszty. Firmy rzadko używają jednego dużego modelu językowego do rozwiązywania wszystkich swoich problemów biznesowych. Natomiast w miarę możliwości przedsiębiorstwa wolą korzystać z mniejszych i tańszych modeli, a duże modele językowe wykorzystywać do skomplikowanych zapytań. Ponadto gdy użycie rośnie, zazwyczaj chcemy spróbować wdrożyć tańsze rozwiązania w zakresie serwowania dużych modeli językowych. Na przykład uruchomienie otwartoźródłowego LLM-a (takiego jak DeepSeek) w egzemplarzu AWS Elastic Compute Cloud (EC2), czyli na serwerze wirtualnym, może być od 30 do 60 procent tańsze niż w przypadku rozwiązania zarządzanego, takiego jak AWS SageMaker. Ponadto często można uzyskać dobrą ofertę (na przykład 70 procent zniżki) w przypadku zarezerwowania egzemplarza EC2, co dodatkowo obniża koszty obsługi.

Typowa aplikacja chatbota może wykorzystywać mniejszy model klasyfikacji intencji do selekcji pytań klientów, model osadzania do kodowania tych pytań i wyszukiwania potencjalnych odpowiedzi oraz duży model językowy w celu podzielenia żądania klienta na akcje (wywołania API backendu) i generowania przyjaznych dla człowieka interakcji z użytkownikiem.

Kolejnym aspektem jest prywatność i bezpieczeństwo danych. Jeśli Twoja firma przetwarza dane wrażliwe lub zastrzeżone, możesz być zmuszony do wykorzystania rozwiązania wewnętrznego, aby uniknąć naruszeń danych lub niezgodności z przepisami.

Większość dostawców modeli bazowych albo nie zapewnia funkcjonalności do serwowania modelu dostrojonego, albo pobiera znacznie wyższą cenę, jeśli musisz obsługiwać własną, dostrojoną wersję modelu. Na przykład w momencie powstawania tej książki na początku 2025 roku

Kup książkę Poleć książkę

firma OpenAI pobiera o 50 procent wyższą opłatę za serwowanie modelu dostrojonego. Jednocześnie mniejsze modele stają się znacznie potężniejsze i umożliwiają dostrajanie do niestandardowych danych i wymagań, szybsze wnioskowanie, lepsze i bardziej spójne wyniki oraz niższe koszty serwowania.

W miarę rozwoju technologii trendy zmierzają w kierunku przystępnych cenowo i otwartych podejść, a rozwiązania oparte na dużych modelach językowych nie są tu wyjątkiem. Nie ma nic złego w korzystaniu z LLM-ów od dostawców, ale posiadanie własnego rozwiązania w zakresie serwowania (w chmurze lub lokalnie) z otwartoźródłowym LLM-em zapewni Twojej firmie istotną przewagę przy niższych kosztach.

Pytanie 3: „Budowanie i utrzymywanie własnego stosu serwującego jest kosztowne. Co mogłoby spowodować, że byłoby to warte swojej ceny?”.

Obniżanie kosztów serwowania modeli to nieustanny wysiłek. Wraz z rozwojem branży oraz zmianami w technologii i infrastrukturze nikt nie może zagwarantować, że proste zlecenie serwowania podmiotowi zewnętrznemu zawsze będzie odpowiadać celom biznesowym. Dokładne zrozumienie koncepcji serwowania modeli może pomóc w poznaniu wszystkich kompromisów, co z kolei pozwoli dokonywać optymalnych wyborów. Przyjrzyjmy się kilku przykładom.

Najpierw wyobraź sobie, że prowadzisz startup w sektorze edukacyjnym. Twoja aplikacja wykorzystuje duże modele językowe do oceniania prac domowych uczniów i generowania spersonalizowanych planów nauki. Współpraca z w pełni zarządzanym dostawcą modelu, takim jak OpenAI, DeepSeek czy Anthropic, to świetny punkt wyjścia do przyspieszenia rozwoju i weryfikacji modelu biznesowego.

Jednak w miarę wzrostu popularności tej aplikacji koszty związane z serwowaniem stają się nie do utrzymania. Aby zoptymalizować wydatki, rozważasz migrację do niestandardowego rozwiązania w zakresie serwowania modeli, na przykład wykorzystanie otwartoźródłowych LLM-ów, wybór zoptymalizowanych frameworków do serwowania i dostrojenie wydajności wnioskowania. Może to znacznie obniżyć koszty, a jednocześnie wciąż zapewniać dobre wrażenia użytkownikom.

Przyjrzyjmy się innemu scenariuszowi. Tym razem prowadzisz firmę zajmującą się zarządzaniem relacjami z klientami (CRM), która wykorzystuje duże modele językowe w celu dostarczania rekomendacji sprzedażowych poprzez przetwarzanie danych z wielu źródeł, w tym danych tekstowych (e-maile, wiadomości z serwisu Slack, notatki klientów), nieustrukturyzowanych wyszukiwań audio (połączenia głosowe, transkrypcje spotkań) i ustrukturyzowanych baz danych CRM (wcześniejsze rekordy sprzedaży, interakcje z klientami).

Biorąc pod uwagę dużą liczbę żądań wnioskowania kierowanych do dużych modeli językowych oraz zaangażowanie danych wrażliwych klientów, pełne zlecenie serwowania modeli podmiotowi zewnętrznemu mogłoby wprowadzić ryzyko związane z zapewnieniem bezpieczeństwa danych i zwiększyć koszty operacyjne. Zatem decydujesz, że konieczne jest posiadanie wewnętrznego stosu serwowania i jego optymalizacja, aby obniżyć koszty związane z wnioskowaniem, zachować zgodność z przepisami dotyczącymi bezpieczeństwa (takimi jak RODO i HIPAA) oraz osiągnąć lepszą wydajność działania i kontrolę opóźnień.

Kup książkę Poleć książkę

Jednocześnie budowa i obsługa wewnętrznego stosu serwowania wymaga znacznych początkowych inwestycji finansowych w specjalistyczny sprzęt, a także ciągłych nakładów na inżynierów z doświadczeniem we wdrażaniu, utrzymywaniu i optymalizowaniu tej infrastruktury. W efekcie decyzja o budowie własnego rozwiązania w porównaniu z outsourcingiem zależy nie tylko od wymagań w zakresie bezpieczeństwa i prywatności, ale także od dojrzałości projektu, jego oczekiwanej skali oraz zdolności organizacji do utrzymania złożoności operacyjnej w czasie.

Z naszego doświadczenia wynika, że chociaż rozwiązania zlecane podmiotom zewnętrznym mogą być świetnym punktem wyjścia, firmy muszą nieustannie oceniać, czy serwowanie modeli we własnym zakresie zapewnia długoterminową przewagę konkurencyjną pod względem oszczędności kosztów, bezpieczeństwa i elastyczności.

Nie ma jednego uniwersalnego rozwiązania w zakresie serwowania modeli

Autorzy książki od ponad dekady budują systemy uczenia maszynowego przeznaczone dla różnych przypadków użycia w przedsiębiorstwach. Pracowaliśmy nad projektami, w których modele w pełni pochodzą od podmiotów zewnętrznych (OpenAI), są samodzielnie zarządzanymi rozwiązaniami lokalnymi (TensorFlow Serving, TorchServe, NVIDIA Triton, Ray), są w pełni zarządzanymi rozwiązaniami dostawców chmury (AWS Bedrock) i dość mocno dostosowanymi do własnych potrzeb rozwiązaniami pochodzącymi od dostawców (AWS SageMaker).

Przez lata nauczyliśmy się, że nie ma jednej uniwersalnej ani wiecznej architektury. Algorytmy modeli i technologie ich serwowania wciąż ewoluują, a firmy muszą dotrzymywać im kroku. Proaktywne wdrażanie nowych technologii pozwala zapewnić lepsze wrażenia użytkownika i obniżyć koszty operacyjne, a jednocześnie pomaga firmie utrzymać się przy życiu i się rozwijać.

Jako właściciel firmy lub programista zajmujący się udostępnianiem modeli musisz mieć rozległą wiedzę w zakresie podstaw serwowania modeli, aby móc zrozumieć innowacje w branży oraz ocenić wady i zalety, ponieważ to pozwala dokonywać właściwych wyborów technicznych.

Koniec końców nie chcemy, abyś był uzależniony od jakichkolwiek konkretnych frameworków lub dostawców — musisz być w stanie szybko zrozumieć i wdrożyć nowe technologie serwowania modeli, aby zapewnić swojej firmie przewagę w obliczu zaciętej konkurencji.

Dlaczego należy optymalizować serwowanie modeli (szczególnie dużych modeli językowych)?

Gdy model zostanie pomyślnie wdrożony, z funkcjonalnego punktu widzenia może wydawać się „gotowy” — generuje poprawne wyniki i odpowiada na żądania. Jednak w przypadku dużych modeli językowych uproszczone wdrożenie systemu serwowania często szybko zawodzi w praktyce. Opóźnienia rosną pod obciążeniem, przepustowość osiąga poziom znacznie poniżej możliwości sprzętu, a koszty serwowania rosną liniowo lub — gorzej — wraz z użyciem. W wielu przypadkach konfiguracja serwowania, która sprawdza się w wersji demonstracyjnej lub

Kup książkę Poleć książkę

pilotażowej, staje się ekonomicznie lub operacyjnie niewykonalna, gdy pojawiają się prawdziwi użytkownicy.

Kolejnym wyzwaniem po hostingu modelu jest optymalizacja kosztów. To rodzi ważne pytania typu: czy możemy uruchomić model na bardziej opłacalnym sprzęcie? Czy możemy poprawić *przepustowość* (liczbę prognoz modelu przetwarzanych w jednostce czasu) i zmniejszyć *opóźnienie* (czas potrzebny na przetworzenie pojedynczej prognozy modelu) bez modernizacji infrastruktury? Czy możemy w pełni wykorzystać usługę LLM dostawcy chmury? W tym miejscu do gry wkracza optymalizacja serwowania modelu.

Optymalizacja serwowania modelu odnosi się do procesu poprawy wydajności serwowania modelu, na przykład zmniejszenia opóźnienia serwowania, zwiększenia przepustowości i optymalizacji wykorzystania zasobów. W praktyce wykorzystujemy optymalizację, aby zmaksymalizować wydajność serwowania przy jednoczesnym utrzymaniu kosztów pod kontrolą.



Optymalizacja jest ważna w przypadku dużych modeli językowych

LLM-y są na tyle duże i złożone, że wymagają znacznej mocy obliczeniowej, co z kolei może oznaczać, że ich koszty operacyjne będą nieakceptowalne dla małych firm. Na przykład w 2023 roku prezes Alphabetu, John Hennessy, powiedział agencji Reuters (*https://www.reuters.com/technology/tech-giants-ai-like-bing-bard-poses-billion-dollar-search-problem-2023-02-22/*), że „wykonanie żądania dużego modelu językowego może być dziesięć razy droższe niż przeprowadzenie tradycyjnego wyszukiwania z użyciem słów kluczowych, co potencjalnie prowadzi do miliardów dodatkowych kosztów".

Tak więc w przypadku dużych modeli językowych optymalizacja serwowania jest niemal koniecznością: pozwala firmom obniżyć koszty, poprawić wrażenia użytkowników dzięki krótszemu czasowi reakcji, a także zyskać przewagę konkurencyjną nad firmami oferującymi wolniejsze lub droższe rozwiązania w zakresie wnioskowania.

W następnym podrozdziale przyjrzymy się przykładowi, który pomoże zrozumieć, dlaczego zoptymalizowany model i system serwowania — zwłaszcza w przypadku LLM-ów — mogą powodować ogromną różnicę w porównaniu z „niezmodyfikowanym" modelem, który pochodzi prosto z operacji trenowania.

Nie martw się, jeśli niektóre z tych pojęć są Ci nieznane! Mamy to pod kontrolą — w rozdziałach od 5. do 8. przybliżymy te koncepcje i wykorzystamy przykłady praktyczne, aby wszystko wyjaśnić, więc prosimy o cierpliwość.

Przykład wykorzystania frameworka do serwowania modeli (vLLM) w celu poprawy przepustowości dużego modelu językowego

Aby dobrze zrozumieć optymalizację serwowania modeli, warto najpierw poznać różnicę między *frameworkiem do uczenia modeli* i *frameworkiem do ich serwowania*. Często zadawane pytanie brzmi: „Dlaczego nie użyć tej samej konfiguracji do serwowania i do uczenia? Przecież oba procesy polegają na uruchamianiu tego samego modelu, prawda?".

Kup książkę Poleć książkę

Formalnie rzecz biorąc, to częściowo jest prawda — oba procesy obejmują wczytywanie modelu i wykonywanie jego architektury. Jednak cele i wymagania są zupełnie inne. Te różnice przedstawia tabela 1.1.

Tabela 1.1. Podstawowe różnice między frameworkami przeznaczonymi do trenowania i serwowania modeli

Aspekt	Trenowanie modelu	Serwowanie modelu
Etap w cyklu życia uczenia maszynowego	Przygotowanie modelu	Wdrożenie do środowiska produkcyjnego
Cel	Uruchomienie modelu jako procesu w celu uczenia się parametrów (wag) poprzez minimalizację funkcji straty na danych treningowych	Uruchomienie modelu w celu wydajnego generowania predykcji (wnioskowania) na nowych danych wejściowych
Obliczenia	Niezwykle intensywne obliczeniowo, obejmujące iteracyjne aktualizacje wag modelu (w tym propagację wsteczną i aktualizacje gradientu)	Skoncentrowanie wyłącznie na wydajnej propagacji w przód (brak propagacji wstecznej i aktualizacji modelu)
Przepustowość i opóźnienie	Preferowane dla wysokiej przepustowości (wiele próbek na sekundę); duże partie danych są zazwyczaj przetwarzane równolegle w celu optymalizacji wykorzystania procesora graficznego	Zoptymalizowane pod kątem niskiego opóźnienia odpowiedzi na żądanie; często operuje na pojedynczych lub małych partiach.
Wymagania dotyczące zasobów	Wymaga potężnych procesorów GPU/TPU i rozproszonych środowisk treningowych (takich jak Fully Sharded Data Parallel [FSDP] i DeepSpeed); przykładem skali jest model Llama3 405B, który był trenowany na 16 384 procesorach graficznych NVIDIA H100	Zoptymalizowane pod kątem niskich opóźnień, oszczędnego wykonywania, często na procesorach CPU, urządzeniach brzegowych lub procesorach graficznych przeznaczonych do wnioskowania (takich jak NVIDIA TensorRT i ONNX Runtime)

Z tabeli 1.1 jasno wynika, dlaczego użycie frameworka do trenowania modeli i systemu do ich serwowania byłoby nieefektywne. Zamiast nich stosujemy środowiska specyficzne dla serwowania modeli, takie jak NVIDIA Triton Inference Server (*https://developer.nvidia.com/dynamo-triton*), vLLM (*https://arxiv.org/pdf/2309.06180*) i SGLang (*https://arxiv.org/abs/2312.07104*).

Teraz przeanalizujmy przykład wnioskowania LLM-a za pomocą frameworka serwowania vLLM, zoptymalizowanego silnika serwowania LLM-ów i zaprojektowanego przez Sky Lab z Uniwersytetu Kalifornijskiego w Berkeley w celu zapewnienia wysokiej przepustowości i niskich opóźnień wnioskowania. Użycie frameworków takich jak vLLM do hostingu LLM-ów może znacząco poprawić wydajność i przepustowość ich serwowania w porównaniu z podejściami bazowymi, takimi jak Hugging Face Transformers lub natywne wykonywanie PyTorch. Te frameworki bazowe są przeznaczone głównie do trenowania modeli.

Podobnie jak inne frameworki serwowania modeli, także vLLM udostępnia inżynierom wiele opcji do dostrojenia w celu uzyskania optymalnej wydajności działania modelu. Spójrz na poniższe polecenie vLLM służące do uruchomienia modelu OpenAI gpt-oss 20b na karcie graficznej H100 80 GB z kilkoma optymalnymi konfiguracjami:

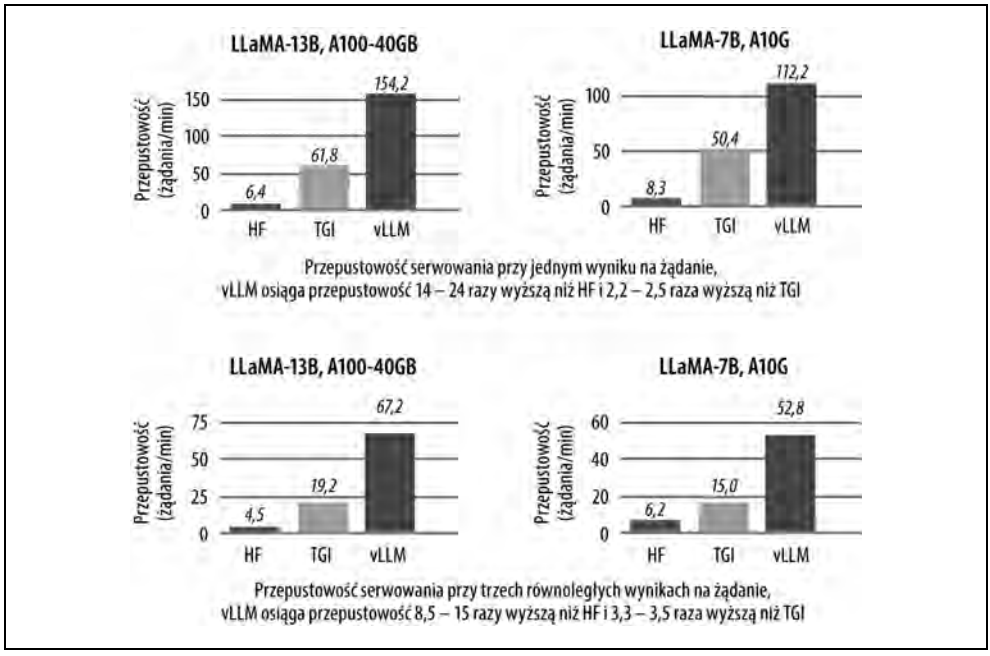
Kup książkę Poleć książkę

```
python -m vllm.entrypoints.openai.api_server \
--model openai/gpt-oss-20b \
--dtype bf16 \
--gpu-memory-utilization 0.9 \
--max-num-seqs 16
--max-num-batched-tokens 16384 \
--tensor-parallel-size 2
```

To polecenie przekazuje do vLLM kilka konfiguracji, które mają na celu osiągnięcie dobrej przepustowości i opóźnienia na GPU NVIDIA H100. Przykładowo określa funkcjonalność PagedAttention (która jest domyślnie włączona w vLLM) jako bufor typu KV, definiuje precyzję modelu na `bf16`, maksymalną liczbę jednoczesnych żądań (wielkość partii danych, `max-nums-seqs`) na 16 i dzieli model na dwa procesory graficzne H100 (`tensor-parallel-size`). Te wszystkie opcje omówimy w rozdziałach 6. i 7., więc nie przejmuj się, jeśli są Ci nieznane.

Przy odpowiedniej konfiguracji wydajność serwowania może w imponującym stopniu się poprawić. W 2023 roku zespół vLLM przeprowadził eksperyment (*https://vllm.ai/blog/2023-06-20-vllm*), który polegał na wdrożeniu PagedAttention w dwóch konfiguracjach: Llama-7B na karcie graficznej z procesorem NVIDIA A10G i Llama-13B na karcie graficznej z procesorem NVIDIA A100 (40 GB).

Zespół osiągnął przepustowość nawet 24-krotnie wyższą niż w przypadku biblioteki HF Transformers (*https://huggingface.co/docs/transformers/main_classes/text_generation*) i do 3,5 raza wyższą niż w przypadku HF Text Generation Inference (TGI) (*https://github.com/huggingface/text-generation-inference*). Rysunek 1.3 ilustruje odkrycia, których dokonano podczas przeprowadzania eksperymentu.



Rysunek 1.3. Biblioteka vLLM osiąga 8,5 – 15 razy wyższą przepustowość niż HF oraz 3,3 – 3,5 raza wyższą przepustowość niż TGI — na podstawie Kwon i in., 2023 (https://vllm.ai/blog/2023-06-20-vllm)

Kup książkę Poleć książkę

Chcemy w tym miejscu podkreślić, jak ogromny wpływ może mieć optymalizacja modelu — zwłaszcza w przypadku dużych modeli językowych. Jeżeli weźmiemy pod uwagę wysokie koszty kart graficznych, wówczas osiągnięcie 6-krotnie większej przepustowości i 3-krotnie mniejszych opóźnień bez żadnych dodatkowych kosztów infrastruktury należy uznać za prawdziwy przełom.

W przypadku produkcyjnych systemów opartych na LLM-ach optymalizacja jest koniecznością i nie chodzi w niej przede wszystkim o wyciśnięcie dodatkowej wydajności, a raczej o spełnienie biznesowych umów o gwarantowanym poziomie świadczenia usług (SLA) dotyczących opóźnień i przepustowości przy jednoczesnym zachowaniu zrównoważonych kosztów.



Konfiguracja frameworka serwującego znacząco wpływa na wydajność działania

Każda opcja dostępna we frameworku serwującym stanowi potencjalną technikę optymalizacji, a odpowiednia konfiguracja może zrobić ogromną różnicę w wydajności.

Na przykład gdy eksperymentowaliśmy z hostingiem modelu DeepSeek R1 za pomocą vLLM, zyskaliśmy 15-krotny wzrost przepustowości — przeskok z 38 do 600 tokenów na sekundę (TPS). Okazało się to możliwe dzięki wprowadzeniu dwóch prostych zmian w konfiguracji vLLM: włączenia jądra FP8 Matrix Learning Accelerator (MLA) i zwiększenia wielkości partii.

Paradygmaty serwowania modeli

W tym podrozdziale przyjrzymy się najpopularniejszym paradygmatom serwowania modeli i ich rzeczywistym zastosowaniom. Te koncepcje będziemy wprowadzać stopniowo — rozpoczniemy od serwowania w urządzeniach oraz serwowania pojedynczych modeli, a następnie przejdziemy do bardziej złożonych architektur serwowania wielu modeli, z których każda bazuje na poprzedniej.

Po drodze podzielimy się spostrzeżeniami ze swoich praktycznych doświadczeń, aby pomóc Ci zrozumieć kompromisy związane z poszczególnymi podejściami. Na zakończenie tego podrozdziału będziesz mieć jasny obraz tego, które rozwiązanie serwowania najlepiej odpowiada Twoim potrzebom biznesowym. Faktyczne zasady projektowania, wraz z praktycznymi przykładami systemów serwowania modeli, omawiamy w rozdziałach 2., 3. i 4.

Serwowanie w urządzeniach (Edge)

Serwowanie w urządzeniu oznacza uruchamianie modelu bezpośrednio w urządzeniach użytkownika, zamiast polegać na zdalnych usługach sieciowych. Taka konfiguracja pozwala aplikacjom przetwarzać dane lokalnie — tam, gdzie są generowane — w urządzeniach takich jak smartfony, drony, roboty, kamery i zestawy VR, a tym samym umożliwia przeprowadzanie obliczeń w czasie rzeczywistym bez konieczności stałego połączenia z internetem.

Kup książkę Poleć książkę

Serwowanie w urządzeniu ma cztery główne zalety:

Opłacalność

Uruchamianie modeli lokalnie w urządzeniach zmniejsza zależność od infrastruktury chmury, co może znacznie obniżyć powtarzające się koszty operacyjne. Na przykład inteligentny głośnik wykonujący rozpoznawanie głosu lokalnie pozwala uniknąć stałych opłat za korzystanie z API w chmurze.

Wydajność

Wykorzystanie obliczeń w urządzeniu przyspiesza czas reakcji i poprawia wydajność energetyczną, zwłaszcza na sprzęcie zoptymalizowanym pod kątem obciążeń AI. Na przykład aplikacje mobilne do ulepszania zdjęć wykorzystujące serwowanie modeli w urządzeniu zapewniają wyniki w czasie rzeczywistym bez opóźnień.

Prywatność

Ponieważ dane nigdy nie opuszczają urządzenia, wnioskowanie w urządzeniu wzmacnia bezpieczeństwo i chroni prywatność użytkowników. Na przykład w przypadku aplikacji do śledzenia stanu zdrowia lokalna analiza osobistych danych biometrycznych pozwala uniknąć przesyłania poufnych informacji na zewnętrzne serwery.

Personalizacja

Modele mogą być stale aktualizowane i dostrajane za pomocą danych specyficznych dla użytkownika bezpośrednio na urządzeniu, bez konieczności wysyłania ich do chmury. Wyobraź sobie na przykład aplikację klawiatury, która z czasem uczy się Twojego stylu pisania, aby usprawnić sugestie — bez przesyłania Twoich naciśnięć klawiszy.

Biorąc pod uwagę te korzyści, serwowanie modeli w urządzeniu zostało szeroko zastosowane w różnych aplikacjach.

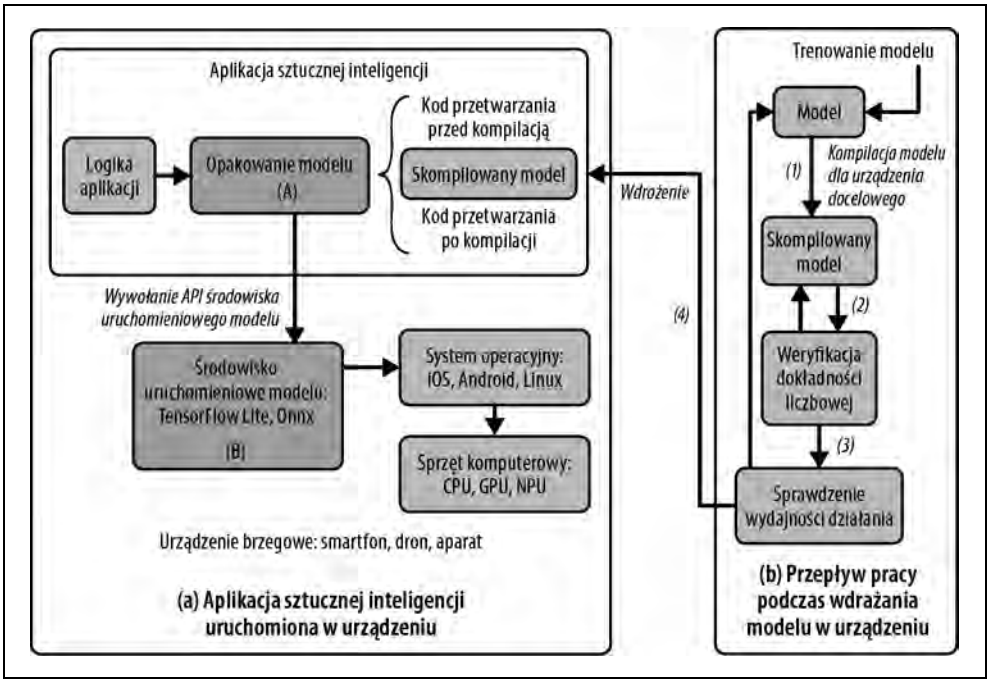
Projektowanie rozwiązania, w którym model jest serwowany w urządzeniu

Teraz przeanalizujemy, jak działa serwowanie modeli w urządzeniach. Rozpoczniemy od przyjrzenia się najważniejszym komponentom aplikacji sztucznej inteligencji, która działa w urządzeniu, a następnie wyjaśnimy proces wdrażania, wymagany w celu przygotowania modelu do działania w różnych urządzeniach. Podstawowe koncepcje ilustruje rysunek 1.4.

Na wysokim poziomie środowisko uruchomieniowe modelu i opakowanie modelu są głównymi komponentami projektu aplikacji sztucznej inteligencji, która ma być uruchomiona w urządzeniu. Zostały one pokazane na rysunku 1.4 (a) jako komponenty A i B.

Środowisko uruchomieniowe modelu to wyspecjalizowany framework oprogramowania zaprojektowany do wydajnego wykonywania modeli uczenia maszynowego na różnych platformach sprzętowych. Służy jako warstwa pośrednia między wytrenowanym modelem i komponentami sprzętowymi urządzenia, optymalizując proces wnioskowania w celu zmaksymalizowania szybkości, wydajności i wykorzystania zasobów.

Kup książkę Poleć książkę



Rysunek 1.4. (a) Projektowanie aplikacji sztucznej inteligencji do działania w urządzeniu i (b) przepływ pracy wdrażania modelu

Środowisko uruchomieniowe modelu odgrywa kluczową rolę w rozwiązaniach sztucznej inteligencji działającej lokalnie na urządzeniu, ukrywając złożoność wykonywania modeli na różnych platformach sprzętowych (takich jak smartfony, drony i roboty) oraz w różnych systemach operacyjnych (iOS, Android, Linux). Zapewnia również optymalizacje specyficzne dla sprzętu i techniki akceleracji w celu poprawy wydajności wnioskowania. Na przykład większość środowisk uruchomieniowych modeli obsługuje tak zwane *delegaty* (*https://developers.google.com/edge/litert/performance/delegates*) — to komponent oprogramowania, który przenosi wybrane operacje wykonywania modelu (zwykle wymagające dużej mocy obliczeniowej) z procesora do wyspecjalizowanego sprzętu, takiego jak procesor graficzny, w celu zoptymalizowania wydajności działania.

Dobrze zaprojektowane środowisko uruchomieniowe modelu pozwala programistom skoncentrować się na tworzeniu funkcji aplikacji, zamiast martwić się o zgodność sprzętową lub szczegóły wykonywania modelu. W chwili powstawania tego tekstu do najpopularniejszych środowisk uruchomieniowych modeli należą LiteRT, ONNX Runtime (ORT) i Core ML.

Jako programiści aplikacji sztucznej inteligencji używamy środowisk uruchomieniowych modeli do ich wydajnego wykonywania i hermetyzacji logiki wykonywania specyficznej dla modelu w moduł wielokrotnego użytku, nazywany *opakowaniem modelu* (komponent A na rysunku 1.4 (a)). To opakowanie ukrywa złożoność szczegółów niskiego poziomu, a tym samym ułatwia integrację, aktualizację i optymalizację modeli w aplikacjach.

Kup książkę Poleć książkę

Opakowanie modelu to komponent zaimplementowany przez programistów aplikacji w celu udostępnienia interfejsów wnioskowania modelu (funkcji) dla logiki aplikacji do wykonania modelu. Opakowanie modelu hermetyzuje szczegóły interakcji ze środowiskiem uruchomieniowym modelu, takie jak wstępne przetwarzanie danych wejściowych modelu, wczytywanie modelu do jego środowiska uruchomieniowego, wykonywanie modelu z jego środowiska uruchomieniowego i przetwarzanie końcowe danych wyjściowych.

W przepływie pracy pokazanym na rysunku 1.4 (a) logika aplikacji najpierw prosi opakowanie modelu o wykonanie modelu wraz z podanymi danymi. Opakowanie modelu zajmuje się konwersją danych i wywołaniem środowiska uruchomieniowego modelu w celu wykonania wskazanego modelu. Następnie środowisko uruchomieniowe modelu obsługuje wykonanie przy użyciu sprzętu lokalnego.

Przygotowanie modelu do serwowania w urządzeniu

Na rysunku 1.4 (b) przedstawiliśmy ogólny przepływ pracy związany z przygotowaniem modelu, od procesu trenowania do działania w urządzeniu.

Po wytrenowaniu modelu należy go najpierw przekonwertować z formatu treningowego na wymagany przez środowisko uruchomieniowe. Na przykład jeśli używasz środowiska uruchomieniowego LiteRT, możesz przekonwertować (skompilować) model Resnet PyTorch do formatu pliku LiteRT (*.tflite*) za pomocą następującego polecenia:

```
ai_edge_torch.convert(resnet18.eval(), sample_input)
```

Po skompilowaniu modelu kolejnym krokiem jest weryfikacja jego dokładności liczbowej w urządzeniu docelowym. Pozwala to upewnić się, że konwersja modelu i optymalizacja pod kątem wykonywania w urządzeniu nie pogorszyły jego dokładności. Najczęstszą metodą weryfikacji jest wykorzystanie tych samych danych wejściowych modelu zarówno w serwerze treningowym, jak i w urządzeniu lokalnym, a potem porównanie różnic w otrzymanych wynikach.

Następnie należy zmierzyć wydajność działania modelu, aby upewnić się, że sprzęt lokalny może go wydajnie używać i jednocześnie spełniać wymagania biznesowe. Po przeprowadzeniu weryfikacji zoptymalizowany model można spakować jako część instalacji lub aktualizacji aplikacji, aby udostępnić go klientom.

Wprawdzie ten proces wdrażania może wydawać się żmudny, ale narzędzia takie jak Qualcomm AI Hub (*https://aihub.qualcomm.com/*) pomagają zautomatyzować wszystkie kluczowe kroki, w tym kompilację, optymalizację i weryfikację modelu w urządzeniu. Dzięki temu przepływ pracy jest znacznie bardziej wydajny.

Kiedy serwowanie w urządzeniu jest właściwym wyborem?

Uruchamianie modeli bezpośrednio w urządzeniach oferuje wiele korzyści, takich jak małe opóźnienia, możliwość działania w trybie offline i zdecydowanie większa prywatność. Oto kilka przypadków użycia, w których preferowany jest hosting modelu w urządzeniu:

Kup książkę Poleć książkę

Prywatność przede wszystkim

Wnioskowanie w urządzeniu okazuje się idealne, gdy nieprzetworzone dane nie mogą opuścić urządzenia użytkownika, na przykład w przypadku uwierzytelniania biometrycznego, danych zdrowotnych lub wrażliwych interakcji z użytkownikiem. Dzięki wykorzystaniu wnioskowania lokalnego systemy mogą spełniać surowe wymagania dotyczące prywatności i zgodności z przepisami bez polegania na usługach zdalnych.

Aplikacje o bardzo małych opóźnieniach

W przypadku aplikacji, w których liczą się milisekundy (rozpoznawanie gestów, pętle sterowania w robotyce, AR/VR lub przetwarzanie dźwięku w czasie rzeczywistym) serwowanie modelu w urządzeniu całkowicie eliminuje opóźnienia sieciowe, a tym samym umożliwia deterministyczne i responsywne działanie.

Środowiska odłączone lub z przerwami w łączności

W scenariuszach z zawodną lub niedostępną łącznością, takich jak sprzęt przemysłowy, zdalne czujniki, pojazdy lub drony, przeprowadzanie wnioskowania w urządzeniu umożliwia nieprzerwane działanie systemu niezależnie od sieci.

Internet rzeczy (IoT), inteligentne miasta i robotyka

Urządzenia brzegowe w inteligentnej infrastrukturze i robotyce są często projektowane z myślą o konkretnych obciążeniach. Modele uruchamiane w urządzeniach są zoptymalizowane pod kątem wydajności energetycznej i specyfiki zadań, co czyni je dobrze przystosowanymi do ciągłego wnioskowania przy ograniczonych zasobach.

Urządzenia lokalne działają w przypadku surowych ograniczeń dotyczących mocy obliczeniowej, zasilania, pamięci masowej i mechanizmów aktualizacji. Te ograniczenia nie sprawiają, że wnioskowanie w urządzeniu jest gorsze — ale silnie wpływają na to, które modele są praktyczne do wdrożenia i kiedy konieczne staje się wnioskowanie hybrydowe lub zdalne. Oto kilka kompromisów i ograniczeń:

Ograniczenia obliczeniowe i pamięci masowej

Smartfony, sprzęt IoT i systemy wbudowane mają ograniczone zasoby CPU, GPU i pamięci, co utrudnia uruchamianie dużych, złożonych modeli. Wprawdzie istnieją modele wytrenowane do działania w urządzeniach brzegowych, takie jak Gemma 3 270M (Google) (*https://developers.googleblog.com/en/introducing-gemma-3-270m/*) i przeznaczone do wdrażania w urządzeniach, ale to nie jest wykonalne dla wszystkich modeli ze względu na ograniczenia pamięci, pamięci masowej i mocy obliczeniowej.

Zużycie energii

Uruchamianie modeli sztucznej inteligencji, zwłaszcza głębokich sieci neuronowych, może być energochłonne i szybko wyczerpywać baterię. Modele przeznaczone do działania w urządzeniach są zazwyczaj projektowane w taki sposób, aby były energooszczędne w zadaniu, do którego zostały zaprojektowane. Mimo to zawsze istnieją granice. Na przykład nie uruchamiamy lokalnie zadania polegającego na ulepszaniu wideo w czasie rzeczywistym (z niskiej do wysokiej rozdzielczości), ponieważ to zużywa nadmierne ilości energii.

Kup książkę Poleć książkę

Ograniczone możliwości w zakresie aktualizacji i konserwacji

Modele w urządzeniach są trudne do częstego aktualizowania. Po wprowadzeniu ulepszeń w modelu każde urządzenie musi otrzymać daną aktualizację i ją zainstalować. Na przykład aplikacja bankowa, która korzysta z uruchamianego w urządzeniu modelu wykrywania oszustw, musiałaby często być aktualizowana, aby poprawić dokładność działania wbudowanego w nią modelu.

Niespójna obsługa sprzętowa

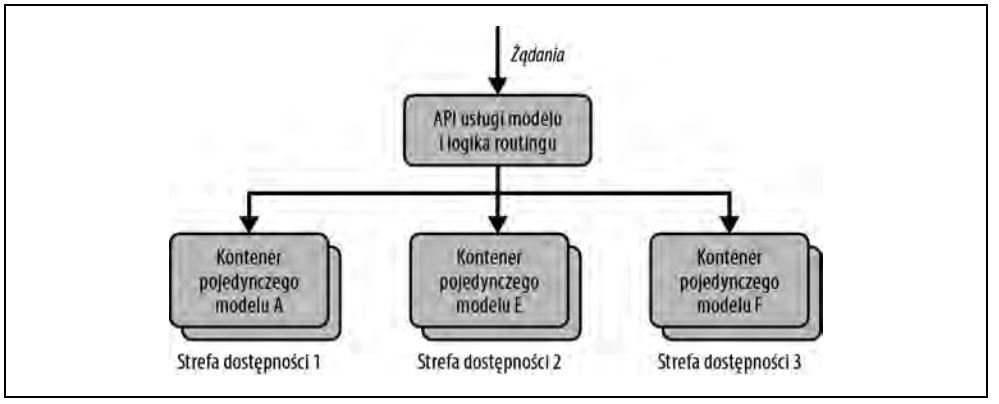
Poszczególne urządzenia mają odmienne możliwości sprzętowe: niektóre obsługują NPU, inne tylko CPU. To może utrudniać wdrożenie zoptymalizowanych modeli we wszystkich urządzeniach. Na przykład model zoptymalizowany pod kątem silnika Neural Engine firmy Apple w iPhonie może nie działać wydajnie w urządzeniu z procesorem Qualcomm Snapdragon i systemem Android.

Gdy ograniczenia związane z działaniem modelu w urządzeniu zaczynają przeważać nad korzyściami — ze względu na wielkość modelu, natężenie ruchu lub złożoność operacyjną — wówczas udostępnianie modelu w naturalny sposób przenosi się do środowisk lokalnych lub chmury. Te środowiska wymieniają lokalne wykonywanie zadań na scentralizowane obliczenia, elastyczne skalowanie i uproszczone operacje. Najczęstszym punktem wyjścia dla udostępniania po stronie serwera jest wzorzec usługi pojedynczego modelu.

Usługa pojedynczego modelu

W tym punkcie przedstawimy klasyczny wzorzec projektowy usługi z pojedynczym modelem, który jest również najczęściej stosowanym podejściem do serwowania modeli w chmurze.

W prezentowanym projekcie (rysunek 1.5) poszczególne modele i ich wersje są wdrażane jako oddzielne usługi sieciowe, udostępniające API przeznaczone do predykcji za pośrednictwem protokołu HTTP lub gRPC w celu wykonania po stronie serwera. Taka konfiguracja zapewnia skalowalność, izolację i elastyczność we wdrażaniu modeli.



Rysunek 1.5. Architektura usługi pojedynczego modelu

Kup książkę Poleć książkę

Jak widać na rysunku 1.5, architektura usługi pojedynczego modelu jest zgodna ze standardowym projektem mikrousług (*https://microservices.io/patterns/microservices.html*). Ma komponent API na froncie, który odbiera z sieci żądania prognoz i kieruje je do jednego z backendowych węzłów roboczych w postaci kontenerów. Następnie te backendowe kontenery przeprowadzają właściwe uruchomienie modelu i zwracają wygenerowany wynik.



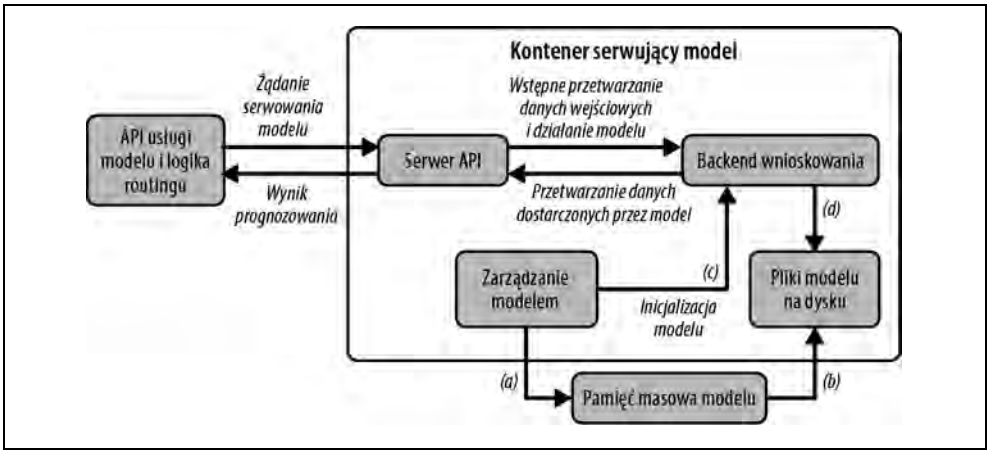
Konteneryzacja jest fundamentem nowoczesnego serwowania modeli

Konteneryzacja oznacza tworzenie odizolowanego środowiska (w tym przypadku *kontenera*), w którym zostaje umieszczona aplikacja wraz ze wszystkimi zależnościami, bibliotekami i konfiguracjami. Użycie kontenera zapewnia spójne działanie aplikacji w różnych środowiskach obliczeniowych, niezależnie od tego, czy to jest laptop programisty, serwer testowy czy środowisko produkcyjne.

W nowoczesnym serwowaniu modeli większość operacji związanych z używaniem modelu po stronie serwera odbywa się wewnątrz kontenera. Powszechną praktyką jest hermetyzacja logiki zarządzania modelem, zarządzania zasobami i wykonywania modelu w kontenerze Dockera (*https://www.docker.com/resources/what-container/*) lub w podzie Kubernetesa (*https://kubernetes.io/docs/concepts/workloads/pods/*) oraz udostępnianie funkcjonalności prognozowania/serwowania modelu za pośrednictwem interfejsu gRPC lub HTTP.

W czasie powstawania książki Docker był najpopularniejszą technologią konteneryzacji, więc w tekście oraz w zamieszczonych przykładach kodu użyliśmy kontenerów Dockera.

Teraz spójrz na rysunek 1.6, aby zobaczyć, co znajduje się wewnątrz kontenera udostępniającego pojedynczy model.



Rysunek 1.6. Projekt kontenera serwującego pojedynczy model

Jak widawć na rysunku 1.6, kontener serwujący model składa się z trzech głównych komponentów: serwera API, zarządzania modelem i backendu wnioskowania. Komponent *backendu wnioskowania* jest odpowiedzialny za działanie modelu. Zwykle wykorzystuje on framework

Kup książkę Poleć książkę

do serwowania modeli (na przykład TensorFlow Serving lub TorchServe) bądź bibliotekę (na przykład vLLM lub TensorRT-LLM) do uruchomienia funkcjonalności serwowania modelu. Z kolei komponent *zarządzania modelem* odpowiada za przygotowanie modelu do serwowania. Obejmuje to pobranie modelu, rozpakowanie go w lokalnej pamięci masowej i wczytanie do backendu wnioskowania (zapoznaj się z krokami inicjalizacji, od (a) do (d), które pokazaliśmy na rysunku 1.6). Natomiast komponent *serwera API* udostępnia funkcjonalność wnioskowania modelu przez HTTP lub gRPC, dzięki czemu aplikacje zewnętrzne mogą przekazywać żądania predykcji.

W przypadku wdrażania modelu oczekujemy, że potok trenowania modelu lub zespół operacyjny przekaże wytrenowany model do pamięci masowej w chmurze. Stamtąd komponent zarządzania modelem wykrywa nowy model i automatycznie odświeża go w każdym kontenerze serwującym, co z kolei zapewnia płynne wdrożenie najnowszej wersji.

Wybór routingu dla usługi serwowania pojedynczego modelu

Jednym z głównych obowiązków komponentu API usługi modelu (rysunek 1.6) jest zagwarantowanie, że wszystkie kontenery serwujące modele będą *w równym stopniu* obciążone zadaniami prognozowania. Wydaje się, że jest to problem, który można łatwo rozwiązać za pomocą strategii rotacyjnego równoważenia obciążenia. Na przykład jeśli masz trzy kontenery (c1, c2 i c3), to pierwsze żądanie prognozowania trafiłoby do c1, drugie do c2, trzecie do c3, a następnie cykl by się powtórzył.

Jednak routing cykliczny ma istotne ograniczenie: nie uwzględnia różnic w czasie przetwarzania żądań. W wielu przypadkach żądania prognozowania mają różne koszty obliczeniowe w zależności od wielkości danych wejściowych i złożoności zadania. Zatem żądanie wnioskowania dużego modelu językowego z większą ilością danych tekstowych (na przykład 5000 tokenów) wymaga dłuższego przetwarzania niż w przypadku danych wejściowych o wielkości 100 tokenów. Model obrazu, który przetwarza obrazy o wysokiej rozdzielczości, zużywa więcej zasobów procesora graficznego lub centralnego niż model przetwarzający mniejsze obrazy.

Cykliczny sposób równoważenia obciążenia powoduje nieefektywność z powodu tej zmienności: na przykład może wystąpić sytuacja, w której kontener c1 nadal przetwarza duże żądanie, podczas gdy kontener c2 już zakończył swoją pracę, a mimo to następne żądanie jest kierowane do c1, co prowadzi do powstania wąskiego gardła.

Aby rozwiązać ten problem, potrzebujemy bardziej wyrafinowanych strategii routingu, które uwzględniają czas przetwarzania, dostępność zasobów i złożoność żądań. Oto kilka często używanych opcji routingu:

Ważony routing cykliczny
: Serwery (większe) o wyższych wagach otrzymują więcej żądań.

Najmniejsza liczba połączeń
: Żądania są kierowane do serwera z najmniejszą liczbą aktywnych połączeń.

Najkrótszy czas odpowiedzi
: Żądania są kierowane do serwera z najkrótszym czasem odpowiedzi.

Kup książkę Poleć książkę

Dynamiczne równoważenie obciążenia

Podczas dystrybucji żądań i ich obsługi wykorzystuje się wskaźniki takie jak użycie procesora, karty graficznej, pamięci operacyjnej lub długość kolejki.

Skalowanie poziome i pionowe

W miarę wzrostu ruchu związanego z żądaniami predykcji pojedynczy kontener serwowania może mieć trudności z szybkim przetwarzaniem żądań. Musimy więc wdrożyć więcej egzemplarzy kontenerów serwujących (będziemy je nazywać po prostu *egzemplarzami*) w wielu maszynach, aby obsłużyć rosnącą liczbę jednoczesnych użytkowników przy zachowaniu minimalnego opóźnienia. Nazywa się to *skalowaniem poziomym* lub *skalowaniem w poziomie*.

W środowisku produkcyjnym obsługującym serwowanie modeli włączamy *automatyczne skalowanie*. Dzięki temu system zarządzania infrastrukturą automatycznie udostępnia więcej egzemplarzy na podstawie stopnia wykorzystania zasobów serwera. System monitoruje kluczowe wskaźniki, takie jak użycie procesora i pamięci, opóźnienie wnioskowania oraz liczba aktywnych żądań.

Gdy monitorowane wskaźniki przekroczą zdefiniowane wartości progowe, system zarządzania infrastrukturą dynamicznie uruchamia dodatkowe egzemplarze. Na przykład jeśli używasz Kubernetesa (*https://kubernetes.io/*) do zarządzania klastrem obliczeniowym, jego mechanizm HPA (ang. *horizontal pod autoscaling*, *https://kubernetes.io/docs/concepts/workloads/autoscaling/horizontal-pod-autoscale/*) może w czasie rzeczywistym, w oparciu o bieżące zapotrzebowanie, automatycznie dostosować liczbę podów serwujących modele.

W przypadku dużych modeli uczenia głębokiego, takich jak GPT-4 i Llama-2-70B, pojedyncza maszyna może nie mieć wystarczającej ilości pamięci graficznej do wczytania modelu. W takich przypadkach stosujemy *skalowanie pionowe* (*skalowanie w górę*). To oznacza wdrożenie modelu z wykorzystaniem potężniejszych procesorów graficznych (takich jak NVIDIA H100 lub A100) i użycie rozwiązania rozproszonego w wielu procesorach graficznych lub maszynach, aby podzielić obciążenie.

Wprawdzie rozproszone serwowanie modeli może być złożonym rozwiązaniem, ale nowoczesne frameworki, takie jak vLLM, upraszczają ten proces poprzez ukrywanie szczegółów niskiego poziomu. Zatem możesz łatwo określić liczbę procesorów graficznych, które mają zostać użyte podczas uruchamiania modelu. Na przykład aby serwować model Llama za pomocą czterech procesorów graficznych i frameworka vLLM, możesz skorzystać z następującego polecenia:

```
python -m vllm.entrypoints.api_server \
   --model meta-llama/Llama-2-13b-hf \
   --tensor-parallel-size 4 \
```

Dlaczego usługa z jednym modelem jest domyślnym punktem wyjścia?

Podejście polegające na serwowaniu pojedynczego modelu jest naszym ulubionym ze względu na jego prostotę i wszechstronność. Działa płynnie z każdym typem modelu, małym lub dużym, od dużych modeli językowych po modele uczenia głębokiego, co czyni je niezawodnym i skalowalnym wyborem w zakresie serwowania modeli.

Kup książkę Poleć książkę



Nadawanie priorytetu serwowaniu wewnątrz węzła (wiele procesorów graficznych w jednym komputerze) względem serwowania między węzłami (z wykorzystaniem wielu komputerów)

W celu uzyskania lepszej wydajności serwowania modeli, łatwiejszego zarządzania i mniejszych opóźnień zazwyczaj lepiej jest uruchamiać duże modele w jednym komputerze wyposażonym w wiele procesorów graficznych (*serwowanie wewnątrz węzła*) niż rozdzielać je na wiele komputerów (*serwowanie między węzłami*).

Serwowanie między węzłami wiąże się z obciążeniem powodowanym przez sieć, co z kolei zwiększa opóźnienie i złożoność synchronizacji. W celu zmaksymalizowania wydajności działania zawsze, gdy to możliwe, zalecamy optymalizację wykonywania modelu, aby zmieścił się w jednym komputerze.

Te techniki optymalizacji modeli przybliżymy w rozdziałach 6. i 7.

W środowisku produkcyjnym i operacyjnym, gdy tylko jest to możliwe, korzystamy z serwowania pojedynczego modelu z następujących powodów:

- Zapewnia najlepszą wydajność ze względu na brak rywalizacji o zasoby i mniejsze opóźnienia.
- Skalowanie jest łatwe, ponieważ każdy model może być skalowany niezależnie.
- Wdrażanie i debugowanie są prostsze dzięki izolowanym dziennikom, wskaźnikom i aktualizacjom.
- Jest bardziej niezawodne, ponieważ awaria jednego modelu nie wpływa na inne.
- Zoptymalizowany sprzęt dla każdego modelu pozwala na lepszą kontrolę kosztów.

Chociaż podejście polegające na serwowaniu pojedynczego modelu jest świetne, ma jednak wady, szczególnie w zakresie wydajności zasobów i wysokości kosztów.

Wyobraź sobie, że prowadzisz platformę agentów, na której klienci mogą tworzyć własne agenty i wdrażać modele niestandardowe. Jeśli masz 100 klientów i każdy z nich wdraża 10 modeli, oznacza to, że Twoja platforma musiałaby zajmować się hostingiem 1000 oddzielnych usług modeli. Taka konfiguracja nie jest skalowalna — koszty ogólne utrzymania, łatania, wdrażania i monitorowania tak wielu usług byłyby przytłaczające. Ponadto wiele przesłanych modeli może nigdy nie zostać użytych, co z kolei prowadziłoby do marnowania zasobów obliczeniowych i ponoszenia niepotrzebnych kosztów związanych z infrastrukturą.

Aby zmniejszyć złożoność operacyjną i zmaksymalizować wykorzystanie zasobów, potrzebujesz bardziej wydajnego rozwiązania — takiego, które pozwala wielu modelom na dynamiczne współdzielenie zasobów obliczeniowych, zamiast uruchamiania oddzielnej usługi dla każdego z nich.

To prowadzi nas do usługi wielomodelowej, czyli podejścia projektowego, które umożliwia opłacalne i skalowalne serwowanie modeli.

Kup książkę Poleć książkę

Usługa wielomodelowa

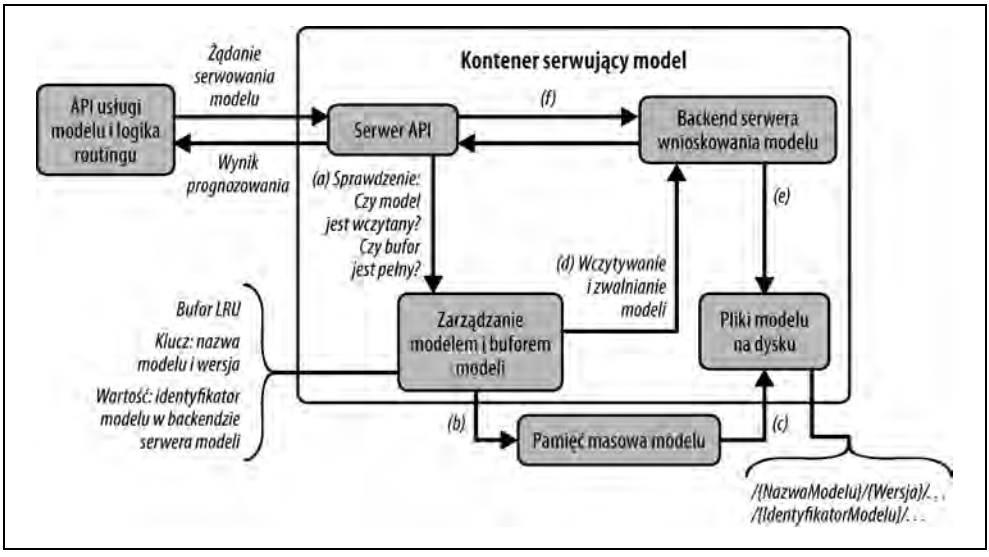
Usługa wielomodelowa została zaprojektowana do hostingu wielu modeli w jednym kontenerze serwującym, współdzielenia procesorów CPU/GPU i pamięci między wieloma modelami oraz dynamicznego wczytywania i zwalniania modeli w zależności od ruchu przychodzącego. Takie rozwiązanie może pomóc w znacznym obniżeniu kosztów i osiągnięciu najlepszego stosunku ceny do wydajności działania.

Usługa wielomodelowa jest idealnym rozwiązaniem, gdy trzeba wydajnie serwować dużą liczbę modeli bez jednoczesnego wczytywania ich wszystkich do pamięci. Zamiast przeznaczać osobny kontener dla każdego modelu, w tym podejściu mamy modele, które *współdzielą* zasoby obliczeniowe w ramach jednego egzemplarza serwującego.

Wracając do naszego przypadku użycia platformy agentów: zamiast jednocześnie wczytywać wszystkie 1000 modeli do procesora graficznego i pamięci, w przypadku usługi wielomodelowej można przyjąć podejście na żądanie. W takim przypadku model jest wczytywany tylko wtedy, gdy klient zażąda predykcji, a zwalniany, gdy jest nieaktywny lub gdy potrzebna jest pamięć dla nowego modelu. Takie podejście znacząco obniża koszty infrastruktury i poprawia wykorzystanie zasobów, a dzięki temu serwowanie modeli jest dużo bardziej skalowalne.

Możesz się zastanawiać: jak to możliwe, aby kontener serwujący zajmował się hostingiem wielu modeli, skoro mogły one zostać wytrenowane przy użyciu różnych frameworków i mogą wymagać różnych parametrów podczas działania?

Odpowiedź leży w projekcie kontenera serwującego modele, który dynamicznie zarządza wczytywaniem, uruchamianiem i zwalnianiem modeli. Przyjrzyjmy się jego architekturze, która została przedstawiona na rysunku 1.7.



Rysunek 1.7. Projekt kontenera serwującego wiele modeli

Kup książkę Poleć książkę

Jeśli porównasz rysunki 1.6 i 1.7 (kontenery serwujące jeden i wiele modeli), zauważysz, że kontener serwujący wiele modeli ma *dwa* komponenty: backend wnioskowania serwera modeli i zarządzania pamięcią podręczną (buforem) modeli.

Backend wnioskowania serwera modeli został zaprojektowany do obsługi wielu typów modeli w sposób przypominający czarną skrzynkę. Niezależnie od algorytmu i wersji modelu backend wnioskowania serwera modeli może udostępniać te modele za pomocą ujednoliconego internetowego API predykcji. Sztuczka polega na tym, że serwer modeli zawiera wewnętrznie wiele typów backendów wnioskowania, z których każdy obsługuje jeden typ modelu, na przykład TensorFlow, ONNX lub PyTorch.

Na szczęście nie musisz budować serwera modeli od zera — dostępne są solidne i uniwersalne rozwiązania otwartoźródłowe, takie jak NVIDIA Triton Inference Server. W praktyce wdrażamy systemy takie jak Triton wewnątrz kontenerów serwujących modele (lub podów Kubernetesa), używając Tritona jako backendu wnioskowania serwera modeli, aby otrzymać wydajne rozwiązanie w zakresie hostingu i zarządzania modelami o różnych wymaganiach, takich jak procesor graficzny, pamięć i frameworki.

Komponent *zarządzania pamięcią podręczną modeli* ma dwa zadania: wczytuje dany model z pamięci masowej do backendu wnioskowania serwera modeli i zapobiega przeciążeniu zasobów kontenera przez modele. W tym celu korzysta z pamięci podręcznej (z algorytmem LRU do usuwania jej elementów) do śledzenia wszystkich modeli i dzięki temu może usunąć z bufora najrzadziej używane modele, gdy stopień wykorzystania zasobów kontenera jest wysoki.



Triton Inference Server — ujednolicona platforma do serwowania wielu modeli

Triton Inference Server to jedno z najpopularniejszych, otwartoźródłowych rozwiązań do serwowania modeli. To również jedno z naszych podstawowych narzędzi, po które warto sięgnąć, zwłaszcza w przypadku zastosowań wielomodelowych.

Triton może zarządzać i serwować dowolną liczbę lub kombinację modeli, ograniczoną jedynie przez miejsce na dysku i zasoby pamięci. Zapewnia ujednolicone API do pracy z różnymi formatami modeli, między innymi TensorRT, TensorFlow GraphDef, TensorFlow SavedModel, ONNX, PyTorch i Caffe2 NetDef. Więcej informacji o tym oprogramowaniu można znaleźć na blogu Triton Architecture (*https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/architecture.html*).

Teraz przyjrzyjmy się kompleksowemu przepływowi pracy w zakresie serwowania modeli na żądanie. Gdy żądanie predykcji w modelu A trafi do kontenera, komponent zarządzania modelem przeprowadzi trzy operacje dla tego żądania:

- Jeśli model A jest już wczytany, komponent zarządzania przekazuje żądanie do backendu serwera modeli, aby uruchomić model i użyć go do wygenerowania wyniku, który następnie zostanie zwrócony.
- Jeśli model A nie jest wczytany, komponent zarządzania pobierze model z pamięci masowej modeli, wczyta go do serwera i przekaże żądanie do backendu wnioskowania serwera modeli.

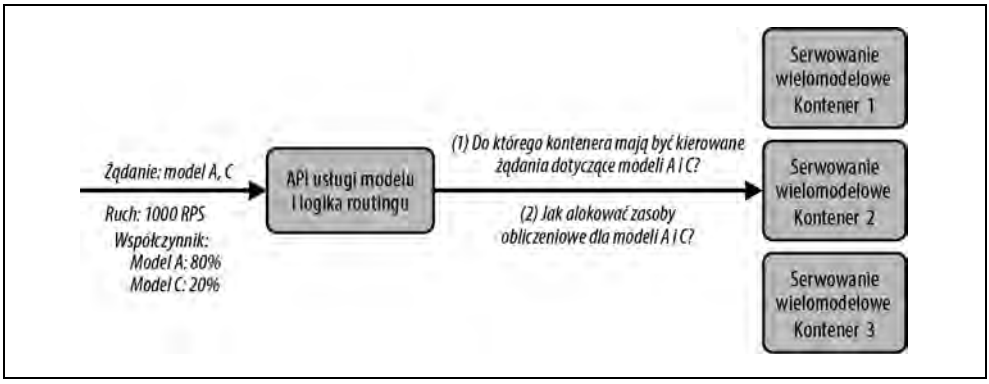
Kup książkę Poleć książkę

- Jeśli stopień użycia pamięci i dysku przekroczy określoną wartość progową (na przykład 80 procent użycia pamięci), to komponent zarządzania znajdzie najrzadziej używany model w swojej pamięci podręcznej, usunie go z backendu serwera, a następnie usunie model z pamięci podręcznej i dysku.

Routing i automatyczne skalowanie w usłudze wielomodelowej

Wprawdzie używanie zasobów współdzielonych na potrzeby hostingu wielu modeli jest wydajne i opłacalne, ale jednocześnie wprowadza pewne wyzwania w zakresie routingu: jak wysyłać różne żądania predykcji modelu do odpowiedniego kontenera serwującego i skutecznie je skalować? Przyjrzyjmy się tym wyzwaniom i przeanalizujmy potencjalne rozwiązanie.

W podejściu przedstawionym na rysunku 1.8 widać dwa problemy. Pierwszym z nich jest routing żądań. Ponieważ modele mogą znajdować się w dowolnym z kontenerów serwujących, chcemy kierować żądanie predykcji do kontenera, który już wczytał model. W przeciwnym razie dodajemy dodatkowe opóźnienie do żądania, wynikające z *zimnego startu* (wstrzymanie żądania w oczekiwaniu na wczytanie modelu do pamięci) lub *wymiany modelu* (usunięcie starych modeli z pamięci w celu jej zwolnienia dla nowego żądanego modelu).



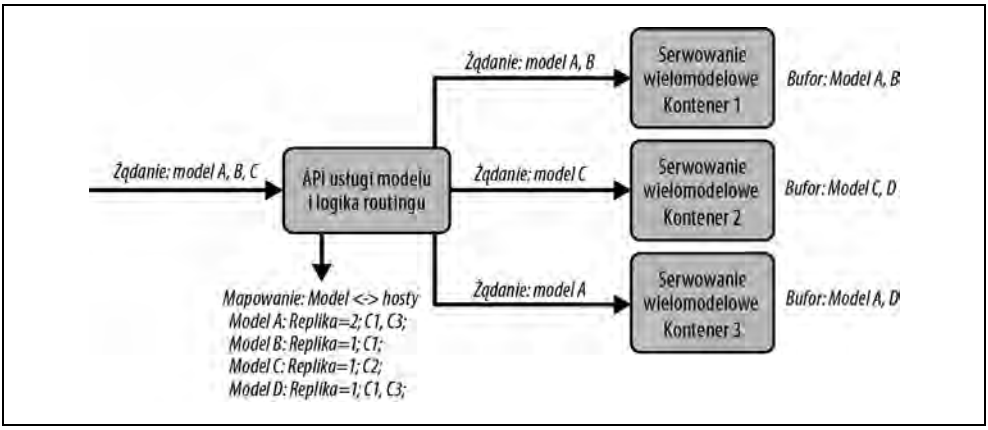
Rysunek 1.8. Wyzwania związane z udostępnianiem i skalowaniem w przypadku serwowania wielu modeli

Drugi problem dotyczy skalowania poszczególnych modeli. Nie wszystkie modele będą obsługiwały taki sam poziom ruchu, niektóre z nich będą używane częściej niż inne. Chcemy mieć więcej egzemplarzy modeli w kontenerach wielomodelowych dla takich „gorących" modeli.

Aby rozwiązać problemy z routingiem i skalowaniem modelu, musimy ulepszyć komponent routingu na warstwie API frontendu (rysunek 1.9).

W rozwiązaniu przedstawionym na rysunku 1.9 dodaliśmy dwie funkcje, aby rozwiązać wspomniane problemy z routingiem i skalowaniem. Po pierwsze dodaliśmy atrybut repliki do metadanych modelu. Wartość repliki modelu określa, ile egzemplarzy tego modelu będzie znajdowało się w kontenerach backendu. Po drugie wprowadziliśmy mapę tras na warstwie routingu i używamy jej do śledzenia, do których kontenerów powinno zostać skierowane dane żądanie predykcji modelu.

Kup książkę Poleć książkę



Rysunek 1.9. Routing żądań serwera modeli za pomocą mapowania host/model i liczby replik modelu

Na przykład na podstawie sekcji `Mapowanie: Model <-> hosty` na rysunku 1.9 dla danego modelu A z liczbą replik wynoszącą 2 logika routingu wybierze dwa kontenery do hostingu modelu A: w tym przypadku będą to kontenery C1 i C3. Następnie komponent routingu będzie równomiernie kierował wszystkie przyszłe żądania modelu A do C1 i C3. Router może również na bieżąco dostosowywać liczbę replik modelu A, zwiększać lub zmniejszać, aby dopasować ją do ruchu obsługiwanego przez model A. Podczas dodawania nowych replik do modelu A zadaniem routera jest również wybranie odpowiedniego kontenera serwującego do hostingu nowo dodanego egzemplarza modelu (przypomina to problem upakowania pojemników, zobacz: *https://en.wikipedia.org/wiki/Bin_packing_problem*).

Kiedy usługa wielomodelowa staje się niezbędna?

Jak już wyjaśniliśmy, serwowanie wielu modeli jest najbardziej opłacalną metodą, ponieważ pozwala wykorzystać wspólną pulę zasobów do hostingu większej liczby modeli. Możliwość uniknięcia hostingu wszystkich modeli w tym samym czasie jest ogromną zaletą.

Jednakże serwowanie wielu modeli ma tendencję do sprawiania problemów w następujących sytuacjach:

- Gdy modele są zbyt duże, aby zmieścić się w jednym procesorze graficznym, gdy nie ma miejsca na współdzielenie albo gdy wymagane jest częste usuwanie modeli w celu zwolnienia miejsca w pamięci. Powoduje to duże obciążenie związane z wczytywaniem modeli i opóźnienia związane z zimnym startem dla przyszłych modeli.
- Gdy poszczególne modele oczekują dużego ruchu, który wymaga małego opóźnienia i ciągłego wczytywania modelu. Wówczas lepsze jest serwowanie pojedynczego modelu.
- Gdy modele mają różne zasady bezpieczeństwa.
- Gdy złożoność operacyjna jest wysoka, na przykład wdrażanie, debugowanie i monitorowanie są skomplikowane ze względu na liczbę ruchomych elementów w systemie, takich jak zarządzanie pamięcią podręczną modeli, routing i skalowanie dla każdego modelu, zgodność modeli i konflikty zależności.

Kup książkę Poleć książkę

Być może zauważyłeś, że podejścia polegające na serwowaniu wielu modeli i jednego modelu uzupełniają się. W praktyce platformy łączą wiele podejść do serwowania, aby obsłużyć różne przypadki użycia pod jednym dachem. W następnym punkcie omawiamy platformy do serwowania modeli.

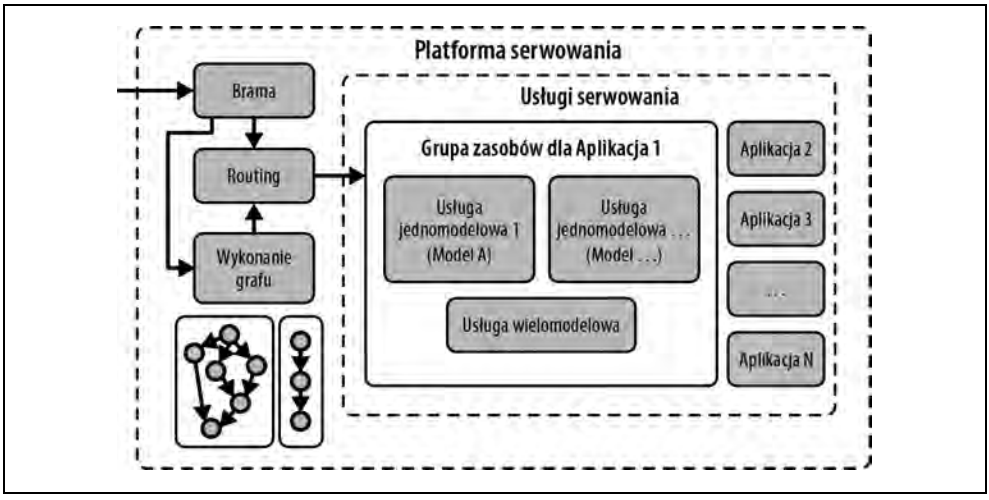
Platformy do serwowania modeli

W miarę rozwoju firmy gwałtownie rośnie zapotrzebowanie na serwowanie modeli, a wymagania stają się coraz bardziej złożone. Zwykłe połączenie usług jednomodelowych i wielomodelowych już nie wystarcza. Systemy serwujące stoją przed dwoma głównymi wyzwaniami.

Po pierwsze wiele zadań wymaga obecnie współpracy wielu modeli. Na przykład asystent głosowy, taki jak Apple Siri, musi zintegrować rozpoznawanie mowy, przetwarzanie języka naturalnego (NLP), modele rekomendacji i syntezę tekstu na mowę, aby odpowiedzieć na jedno polecenie głosowe. Wydajne zarządzanie tymi wielomodelowymi interakcjami ma kluczowe znaczenie dla płynnego działania całości.

Po drugie gdy na jednej platformie działa coraz więcej aplikacji sztucznej inteligencji, optymalizacja zasobów obliczeniowych staje się wyzwaniem o znaczeniu krytycznym. Potrzebna jest wydajna strategia przydzielania procesorów graficznych, procesorów i pamięci między modelami, a jednocześnie zapewnienie skalowalności i opłacalności oraz minimalizacji ryzyka, że dojdzie do rywalizacji o zasoby.

Pokazany na rysunku 1.10 projekt platformy serwującej modele daje wyobrażenie o tym, jak platforma serwująca modele radzi sobie z wyżej wymienionymi wyzwaniami. Przede wszystkim wykorzystuje grupy zasobów do segregacji obciążenia predykcyjnego dla różnych aplikacji i scenariuszy biznesowych. Na przykład w grupie zasobów App1 skonfigurowano dwie usługi jednomodelowe i jedną usługę wielomodelową. Każdej grupie zasobów przydzielane są własne limity procesora, procesora graficznego i pamięci, które opierają się na wymaganiach biznesowych w celu optymalizowania wydajności i kosztów.



Rysunek 1.10. Projekt platformy serwowania modeli

Kup książkę Poleć książkę

Po drugie projekt zawiera komponent wykonujący grafy, na przykład Airflow (*https://airflow.apache.org/*) lub Ray (*https://www.ray.io/*), aby obsługiwać wieloetapowe przepływy pracy wnioskowania. Każdy zespół ds. aplikacji definiuje własny przepływ pracy wnioskowania i wdraża go w silniku wykonywania grafów. W czasie predykcji silnik grafów najpierw pobiera żądanie i uruchamia predefiniowany przepływ pracy wnioskowania. Dla każdego kroku wnioskowania w przepływie pracy silnik grafów wywołuje komponent routingu, aby znaleźć odpowiednią grupę serwowania i usługę do uruchomienia wnioskowania modelu.

Diagram na rysunku 1.10 przedstawia ogólny przegląd platformy serwującej modele. W praktyce ten projekt mógłby być znacznie bardziej skomplikowany. Na przykład platforma serwująca musi mieć komponenty bezpieczeństwa i kontroli dostępu, komponenty wskaźników i monitorowania, zapewniać integrację z rozwiązaniami wdrażania i DevOps oraz obsługiwać wiele innych zadań. Ponieważ te komponenty nie są kluczowe dla serwowania modeli, pominęliśmy je na tym diagramie, aby skoncentrować się na głównych koncepcjach serwowania.

Podsumowanie

W rozdziale przeanalizowaliśmy strukturę modelu z inżynierskiego punktu widzenia, traktując go jako wykonywalne komponenty, a nie tylko statyczne pliki danych. Model składa się z trzech głównych elementów: danych modelu (wag, obciążeń i konfiguracji), architektury modelu (struktury warstw i operacji modelu) oraz kodu wykonywalnego modelu (który wczytuje i uruchamia model w celu wnioskowania).

Następnie wprowadziliśmy pojęcie serwowania modeli, które polega na wdrażaniu modeli w środowiskach produkcyjnych w celu przetwarzania danych w czasie rzeczywistym i generowania predykcji. Zwróciliśmy uwagę na trzy scenariusze wdrożenia — serwowanie modeli w urządzeniu, lokalnie i w chmurze — oraz nakreśliliśmy najistotniejsze kwestie związane z serwowaniem modeli, takie jak skalowalność, opóźnienie, monitorowanie, bezpieczeństwo i optymalizacja kosztów.

Następnie wyjaśniliśmy, dlaczego zrozumienie i optymalizacja serwowania modeli są niezbędne, nawet w przypadku korzystania z rozwiązań dostawców usług chmury lub dużych modeli językowych. Chociaż serwowanie modeli w chmurze jest wygodne, firmy nadal muszą radzić sobie ze złożonymi integracjami, kompromisami kosztowymi i kwestiami bezpieczeństwa. wwych, możliwości dostrajania i opłacalnych strategii wdrażania, aby były praktyczne w rzeczywistych zastosowaniach.

Wprowadziliśmy również optymalizację serwowania modeli, szczególnie w przypadku dużych modeli językowych, gdzie wzrost wydajności może prowadzić do znacznych redukcji kosztów i poprawy wydajności. Optymalizując techniki takie jak pamięć podręczna typu KV z Paged Attention (vLLM) i Radix Attention (SGLang), firmy mogą osiągnąć sześciokrotnie wyższą przepustowość i zmniejszyć opóźnienie o dwie trzecie bez ponoszenia dodatkowych kosztów związanych z infrastrukturą. Ponieważ wnioskowanie przez LLM-y jest kosztowne obliczeniowo, optymalizacja serwowania nie jest tylko ulepszeniem — jest koniecznością dla opłacalnego wdrożenia sztucznej inteligencji.

Kup książkę Poleć książkę

Na koniec przeanalizowaliśmy najpopularniejsze paradygmaty serwowania modeli, począwszy od serwowania w urządzeniu (brzegowego) po platformy serwowania pojedynczych modeli, wielu modeli i pełnych modeli. Każdy paradygmat ma swoje mocne strony i kompromisy, w zależności od czynników takich jak opóźnienie, koszt, skalowalność i złożoność.

Mamy nadzieję, że ten rozdział przekonał Cię, że serwowanie modeli ma znaczenie kluczowe — i że to jest dyscyplina inżynierska, w której koncentrujemy się na wydajnym wdrażaniu i integrowaniu modeli z rzeczywistymi aplikacjami, a nie na zawiłościach trenowania modeli czy projektowania algorytmów.

W kolejnych dwóch rozdziałach zagłębimy się w przykłady praktyczne, aby zobaczyć, jak serwowanie modeli jest budowane i działa w praktyce. Zaczniemy od serwowania dużych modeli językowych w rozdziale 2., a następnie przejdziemy do szerszego projektowania i wdrażania systemów serwowania modeli w rozdziale 3.

Kup książkę Poleć książkę

Kup książkę Poleć książkę


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

Wdrażaj i optymalizuj modele LLM szybko, tanio i na dużą skalę

Rewolucja sztucznej inteligencji sprawiła, że duże modele językowe (LLM) z ciekawostek badawczych stały się krytyczną infrastrukturą produkcyjną. Obecnie największym wyzwaniem dla firm IT nie jest już samo trenowanie modeli, ale ich wydajne, bezpieczne i opłacalne wdrażanie. W erze agentów AI i w obliczu rosnących kosztów korzystania z publicznych API organizacje coraz częściej decydują się na hosting własnych rozwiązań. Ta książka stanowi kompleksowy przewodnik po serwowaniu i optymalizacji LLM. Wypełnia lukę między teorią a praktyką, pokazując, jak przekuć potężne modele w niezawodne aplikacje działające w czasie rzeczywistym, a przy tym poradzić sobie z nową fizyką i ekonomią generatywnej sztucznej inteligencji.

Ta publikacja krok po kroku przeprowadza czytelnika przez cykl życia modelu w środowisku produkcyjnym. Autorzy demistyfikują architekturę transformerów, szczegółowo omawiając fazy wstępnego wypełniania (prefill) i dekodowania, jak również zarządzanie pamięcią podręczną KV. Książka uczy projektowania systemów dla pojedynczych i wielu modeli, a także wdrażania zaawansowanych technik optymalizacji: ciągłego przetwarzania wsadowego, kwantyzacji, dekodowania spekulatywnego czy buforowania prefiksów. Czytelnik dowie się, jak skutecznie używać nowoczesnych frameworków (w tym vLLM, TensorRT-LLM, SGLang, llama.cpp) i jak skalować infrastrukturę na wiele procesorów graficznych (GPU) i węzłów przy zachowaniu idealnego balansu między przepustowością, opóźnieniami a kosztami operacyjnymi.

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 978-83-289-4204-2
9 788328 942042
Cena: 99,00 zł

Najważniejsze zagadnienia

- Architektura LLM i mechanizmy generowania tokenów
- Zarządzanie pamięcią KV i ciągłe przetwarzanie wsadowe
- Projektowanie systemów serwowania
- Zaawansowana optymalizacja
- Skalowanie na wiele GPU
- Przegląd frameworków: vLLM, TensorRT-LLM, SGLang, llama.cpp

Chi Wang — dyrektor do spraw inżynierii w dziale Einstein AI w Salesforce. Ma ponad 18-letnie doświadczenie w zakresie AI i systemów rozproszonych. Kieruje rozwojem wielkoskalowych platform AI, jest autorem 12 patentów.

Peiheng Hu — inżynier specjalizujący się w inferencji LLM w NVIDIA. Absolwent Harvardu i Georgia Tech. Od ponad dekady tworzy rozproszone systemy AI. Wcześniej pracował dla Salesforce i Microsoft Azure.

Gdy duże modele językowe stają się podstawą nowoczesnych platform sztucznej inteligencji, ta książka oferuje praktyczny plan serwowania i optymalizacji modeli LLM na dużą skalę — co umożliwia niezawodną i opłacalną obsługę ogromnych obciążeń.

Bhavesh Doshi, wicedyrektor działu chmury AI w Salesforce