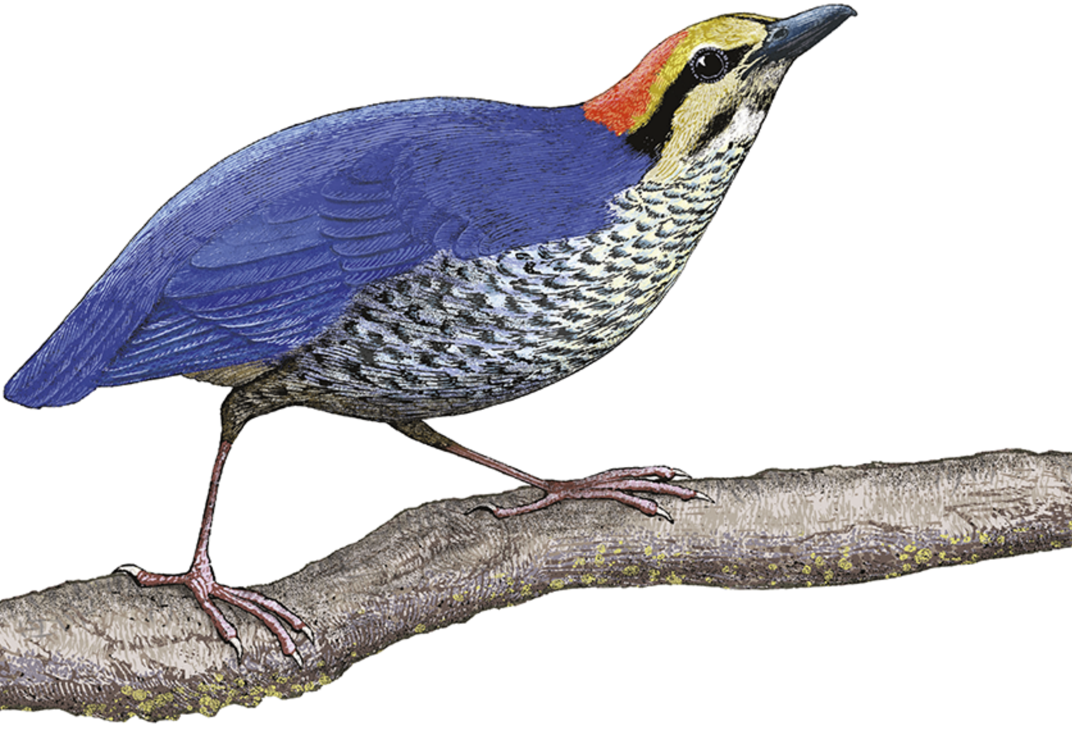
O'REILLY

Wzorce projektowe w generatywnej AI

Sprawdzone rozwiązania przy budowie agentów i aplikacji GenAI



Valliappa Lakshmanan
Hannes Hapke

Helion

Tytuł oryginału: Generative AI Design Patterns: Solutions to Common Challenges
When Building GenAI Agents and Applications

Tłumaczenie: Radosław Meryk

ISBN: 978-83-289-3820-5

© 2026 Helion S.A.

Authorized Polish translation of the English edition of *Generative AI Design Patterns*
ISBN 9798341622661 © 2026 Valliappa Lakshmanan and Hannes Hapke.

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/wzprge
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

ROZDZIAŁ 1.

Wprowadzenie

Generatywna sztuczna inteligencja (GenAI) jest tak potężna i tak łatwa w użyciu, że nawet osoby bez przygotowania technicznego potrafią szybko tworzyć prototypy bardzo przekonujących aplikacji opartych na GenAI. Przeniesienie takich prototypów do środowiska produkcyjnego jest jednak trudne, ponieważ modele GenAI nie są niezawodne — czasami halucynują, zwracają różne odpowiedzi na ten sam prompt i charakteryzują się zaskakującymi ograniczeniami wynikającymi ze sposobu, w jaki zostały wytrenowane. Wzorce projektowe opisane w tej książce stanowią zbiór sprawdzonych praktyk i rozwiązań dotyczących wyżej wymienionych oraz innych powtarzalnych problemów, które niemal na pewno napotkasz podczas budowania produkcyjnych aplikacji opartych na modelach GenAI.

Wzorce projektowe GenAI

W inżynierii oprogramowania *wzorce projektowe* to sprawdzone rozwiązania typowych problemów, które pojawiają się podczas projektowania i programowania systemów. Opisują ustandaryzowane dobre praktyki, wypracowane przez lata dzięki zbiorowemu doświadczeniu twórców oprogramowania. Wzorce projektowe mają znaczenie, ponieważ dają programistom wspólny język, który pozwala sprawnie się komunikować, a przy okazji pomagają podnieść jakość, ułatwiają utrzymanie systemów i wspomagają ich skalowanie.

Na sposób myślenia o wzorcach projektowych ogromny wpływ wywarła praca architekta Christophera Alexandera, który wprowadził pojęcie „wzorców” w architekturze w książce *A Pattern Language* (Oxford University Press, 1977). Wzorce projektowe zyskały szczególną popularność w inżynierii oprogramowania po publikacji książki *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku* (Helion) autorstwa Ericha Gammy, Richarda Helma, Ralpha Johnsona i Johna Vlissidesa, znanej powszechnie jako „książka Gangu Czterech”. Od tego czasu powstały katalogi wzorców dla kolejnych obszarów inżynierii oprogramowania — na przykład dla aplikacji Java Enterprise czy uczenia maszynowego.

Dziś, gdy powstają produkty AI, programiści, zamiast budować własne modele ML, które trzeba trenować od podstaw na danych specyficznych dla aplikacji, coraz częściej opierają się na *bazowych* modelach GenAI (takich jak GPT-4, Gemini, Claude, Llama, DeepSeek, Qwen czy Mistral), wytrenowanych na ogromnych, ogólnych zbiorach danych. W tej książce — za Chip

Huyen i jej *Inżynierią AI. Tworzenie aplikacji z wykorzystaniem modeli bazowych* (Helion) — będziemy nazywać takie podejście *inżynierią AI*, a osoby pracujące w tym paradygmacie — *inżynierami AI*.

Inżynieria AI ma bardzo szerokie zastosowanie — obejmuje między innymi przetwarzanie języka naturalnego (ang. *natural language processing* — NLP), generowanie tekstu, wyjaśnianie kodu, rozumienie obrazów i syntezę wideo. Dzięki temu pozwala obsługiwać takie przypadki użycia jak generowanie treści, asystentów AI, automatyzację przepływów pracy czy robotykę.

Inżynier AI może poprosić model bazowy o bezpośrednie wygenerowanie treści potrzebnych aplikacji. Wystarczy, że wyśle do modelu odpowiednie tekstowe dane wejściowe, czyli *prompt*. W praktyce szybko natrafisz na typowe problemy — wygenerowana treść może nie pasować do oczekiwanego stylu, może nie uwzględniać wiedzy firmowej, której model nie zna, albo może nie mieć pewnych zdolności. W tej książce porządkujemy zestaw sprawdzonych rozwiązań takich problemów, które pojawiają się przy budowaniu aplikacji opartych na bazowych modelach GenAI.

Znajdziesz tu także szczegółowe omówienia 32 wzorców, które przekładają osiągnięcia badań i doświadczenie ekspertów na wskazówki gotowe do zastosowania w projektach. Każdy rozdział przedstawia zestaw wzorców jako potencjalne rozwiązania konkretnego problemu, który często pojawia się w inżynierii AI. Na przykład rozdział 3. dotyczy problemu modeli bazowych polegającego na braku umiejętności generowania treści uwzględniających poufne dane przedsiębiorstwa ze względu na to, że dostawcy modeli trenują je bez dostępu do tych danych. Wszystkie wzorce opisane w tym rozdziale odpowiadają właśnie na to wyzwanie. Każdy podrozdział poświęcony wzorcowi zawiera opis problemu, sprawdzone rozwiązanie, kompletny działający przykład „od początku do końca” oraz omówienie alternatyw i dodatkowych kwestii dotyczących implementacji.

Inżynierowie AI często są zmuszeni do rozwiązywania zadań, które są zbyt złożone na to, aby model bazowy, mógł wykonać je w odpowiedzi na jeden prompt. W takiej sytuacji typowa strategia polega na podziale zadania na mniejsze elementy, które modele bazowe potrafią zrealizować osobno. Takie niewielkie komponenty programistyczne, które zapewniają funkcje dzięki wykorzystaniu modeli bazowych, nazywa się *agentami*. Autonomia agentów rośnie, gdy wykorzystują modele GenAI do zaplanowania sekwencji działań, wyboru narzędzi backendowych dla każdego kroku, odzyskiwania kontroli po błędach i oceny, czy zadanie zostało zakończone. Aplikacje tworzone przez orkiestrację agentów nazywa się *agentowymi* (ang. *agentic*). Wzorce opisane w tej książce, które pokazują, jak radzić sobie z nieuniknionymi wyzwaniami przy budowaniu rozwiązań opartych na modelach bazowych, pomogą Ci budować lepszych agentów i lepsze aplikacje agentowe.

Budowanie aplikacji na podstawie modeli bazowych

W tym podrozdziale krótko przedstawimy podstawy inżynierii AI. Dzięki temu unikniemy konieczności powtarzania tego wprowadzenia w dalszych rozdziałach, poświęconych konkretnym wzorcom. Osobom, które oczekują dokładniejszego omówienia budowy aplikacji

GenAI, polecamy pozycje, takie jak: Omar Sanseviero i współautorzy *Hands-On Generative AI with Transformers and Diffusion Models* (O'Reilly), która wyjaśnia technologię bazową; Chris Fregly i współautorzy *Generatywna sztuczna inteligencja na platformie AWS* (Helion), omawiająca ofertę dostawców chmury na dużą skalę (ang. *hyperscalers*); oraz Leonid Kuligin i współautorzy *Generative AI on Google Cloud with LangChain* (Packt), która opisuje framework open source GenAI oparty na LangChain.

Uwaga o modelach i frameworkach

W książce *Machine Learning Design Patterns*, aby zachować spójność, użyliśmy tylko dwóch frameworków (`scikit-learn` i `TensorFlow`) oraz jednego dostawcy chmury (Google Cloud Platform, GCP). Wielu czytelników wtedy jednak uznało, że przykłady zbyt mocno faworyzowały TensorFlow i GCP. Dlatego w tej książce staramy się zachować neutralność względem modelu, frameworka i dostawcy chmury.

Przykłady kodu obejmują szeroki zestaw technologii pochodzących od różnych dostawców: duże modele językowe (LLM) firm OpenAI, Anthropic, Google, Alibaba i Meta oraz frameworki GenAI, takie jak LangChain, Pydantic AI, Hugging Face i DSPy. Przykłady pozostają również niezależne od dostawców chmury publicznej, takich jak Amazon Web Services (AWS), Azure, GCP czy Oracle Cloud Infrastructure. Najpewniej zastosujesz inny model w odmiennym frameworku oraz scenariuszu, dlatego kod ma pełnić wyłącznie rolę punktu wyjścia. W praktyce konieczne będzie jego dostosowanie do wybranego LLM, frameworka i dostawcy chmury.

Prompt i kontekst

Gdy budujesz aplikacje AI, zwykle wywołujesz hostowane modele bazowe przez API. Może to być API dostawcy modelu, ale równie dobrze może to być framework, który ułatwia przełączanie się między dostawcami.

Model bazowy wywołujesz poprzez wysłanie *promptu*, a w odpowiedzi dostajesz *odpowiedź*. Znasz to zapewne z interfejsu WWW takiego narzędzia. Na przykład w ChatGPT (*http://chatgpt.com*) możesz wpisać do pola tekstowego prompt w rodzaju:

> Utwórz szkic ołówkiem w stylu Degasa, przedstawiający czteroosobową rodzinę grającą w grę planszową.

Najprostszy prompt to zwykle *instrukcja*, która prosi model o wykonanie zadania generowania treści. W tym przypadku model realizuje polecenie i odsyła odpowiedź zawierającą obraz zgodny z opisem (zob. rysunek 1.1)[1]. Zarówno prompty, jak i odpowiedzi mogą mieć charakter *multimodalny* — mogą to być teksty, ale także obrazy, wideo lub dźwięk.

---

[1] Przyjrzyj się uważnie szkicowi. Czy widzisz jakieś problemy? Czy wszystkie postacie mają właściwą liczbę palców? Czy twarz dziewczynki jest symetryczna? Czy gra planszowa przedstawia coś rozpoznawalnego? W chwili pisania tej książki modele generujące obrazy nadal mają kłopot z symetrią, liczeniem oraz wiernym odtwarzaniem szczegółów.

Utwórz szkic ołówkiem w stylu Degasa, przedstawiający czteroosobową rodzinę grającą w grę planszową.

Rysunek 1.1. Wysłanie promptu do modelu bazowego i uzyskanie odpowiedzi

Bardziej złożony prompt może zawierać nie tylko instrukcję, lecz także *kontekst*. Może to być informacja, którą model ma uwzględnić podczas generowania, albo prośba o przyjęcie określonej roli. Na przykład możemy poprosić Claude'a (na Claude.ai), aby przyjął konkretną osobowość i wykorzystał podane informacje przy tworzeniu treści:

> Jesteś ekspertem w dziedzinie marketingu i świetnie znasz rynek książki w niemieckich miastach uniwersyteckich.
>
> *Covenant of Water* to powieść opowiadająca historię trzech pokoleń prawosławnej rodziny chrześcijan św. Tomasza z Kerali.
>
> Napisz jednoakapitową notkę polecającą, która przedstawi książkę czytelnikom w księgarni w Getyndze, nawiązującą do lokalnego kontekstu.

Odpowiedź Claude Sonnet uwzględnia rolę, kontekst i instrukcję:

Covenant of Water — literacka podróż do Kerali

Pośród historycznych murów Getyngi, gdzie tradycja akademicka spotyka się z literacką ciekawością, przedstawiamy mistrzowski epos Abrahama Verghesego *The Covenant of Water*. Ta rozległa saga prowadzi przez losy trzech pokoleń prawosławnej rodziny chrześcijan św. Tomasza z Kerali w Indiach — regionu o złożonym dziedzictwie religijnym, które może wybrzmieć szczególnie mocno wśród studentów słynnych getyńskich studiów teologicznych. Gdy opowieść rozciąga się na kolejne dekady, wśród bujnych rozlewisk Kerali, czytelnicy znający związek Getyngi z wodą — od fontanny Gänseliesel po meandrującą rzekę Leine — dostrzegą zaskakujące analogie w sposobie, w jaki geografia wpływa na ludzkie przeznaczenie. Verghese splata wątki rodzinnych tajemnic, medycznych zagadek oraz więzi międzypokoleniowych, które przekraczają granice kultur. W efekcie międzynarodowa wspólnota badaczy i czytelników otrzymuje pogłębioną refleksję nad tym, jak woda — zarówno w monsunach Kerali, jak i w łagodnych deszczach Dolnej Saksonii — pełni rolę metafory i przymierza oraz spaja doświadczenia ludzi ponad czasem i przestrzenią.

Wywoływanie modelu przez API dostawcy

API do wywoływania modeli bazowych są dostępne dla wielu języków — powszechnie używa się Pythona, Go, Javy i TypeScriptu. Dla uproszczenia w tej książce skoncentrujemy się na Pythonie.

Aby wywołać Claude Sonnet 3 przez API Anthropica w Pythonie, możesz użyć pakietu Anthropic Python w następujący sposób (pełny kod znajduje się na GitHubie — *https://oreil.ly/NauRN*):

```
import anthropic
client = anthropic.Anthropic(
    # defaults to os.environ.get("ANTHROPIC_API_KEY")
    api_key="YOUR_ANTHROPIC_API_KEY",
)
completion = client.messages.create(
    model="claude-3-7-sonnet-latest",
    system="You are an expert Python programmer.",
    messages=[
        {
            "role": "user",
            "content": [
               {
                  "type": "text",
                  "text": """
Write code to find the median value of a list of integers."""
               }
            ]
        }
    ]
)

print(completion.content[0].text)
```

W tym wywołaniu API prompt został podzielony na dwie części — *prompt systemowy* i *prompt użytkownika*. Prompt systemowy ustawia programista aplikacji. Ten prompt decyduje o ogólnym zachowaniu modelu, natomiast prompt użytkownika jest bardziej dynamiczny i obejmuje konkretne instrukcje dla danego zadania. W tym przykładzie rolę asystenta ustawiono w prompcie systemowym, natomiast prompt użytkownika i kontekst przekazano jako wiadomości.

Framework niezależny od dostawcy LLM

To samo zadanie wykonasz we frameworku PydanticAI. Możesz użyć kodu w rodzaju poniższego (zakładamy, że wymagany klucz API jest zapisany w zmiennej środowiskowej):

```
from pydantic_ai import Agent
agent = Agent('anthropic:claude-3-7-sonnet-latest',
              system_prompt="You are an expert Python programmer.")

result = agent.run_sync(
             "Write code to find the median value of a list of integers.")
print(result.data)
```

Zaletą tego podejścia jest możliwość łatwego przełączania się między dostawcami modeli bazowych — wystarczy zmienić identyfikator modelu na `openai:gpt-4o-mini`, `googlevertex:gemini-2.0-flash`, `groq:llama3-70b-8192` i podobne (pełną listę obsługiwanych modeli znajdziesz w dokumentacji biblioteki Pydantic: *https://oreil.ly/Kg6jE*).

Klasa w API Pydantic, która wywołuje model Claude, nosi nazwę `Agent`. Czym są agenty, opiszemy w kolejnym podrozdziale, natomiast teraz dokończymy omówienie sposobów wywoływania modeli bazowych.

Uruchamianie modelu lokalnie

Aby uruchomić model taki jak Llama 3 na własnym sprzęcie, możesz skorzystać z klienta Ollama (*https://ollama.com*), pobrać model i uruchomić go poleceniem:

```
ollama run llama3.2
```

Ollama udostępnia modele o otwartych wagach przez API zgodne z OpenAI, więc możesz użyć, na przykład, następującego kodu:

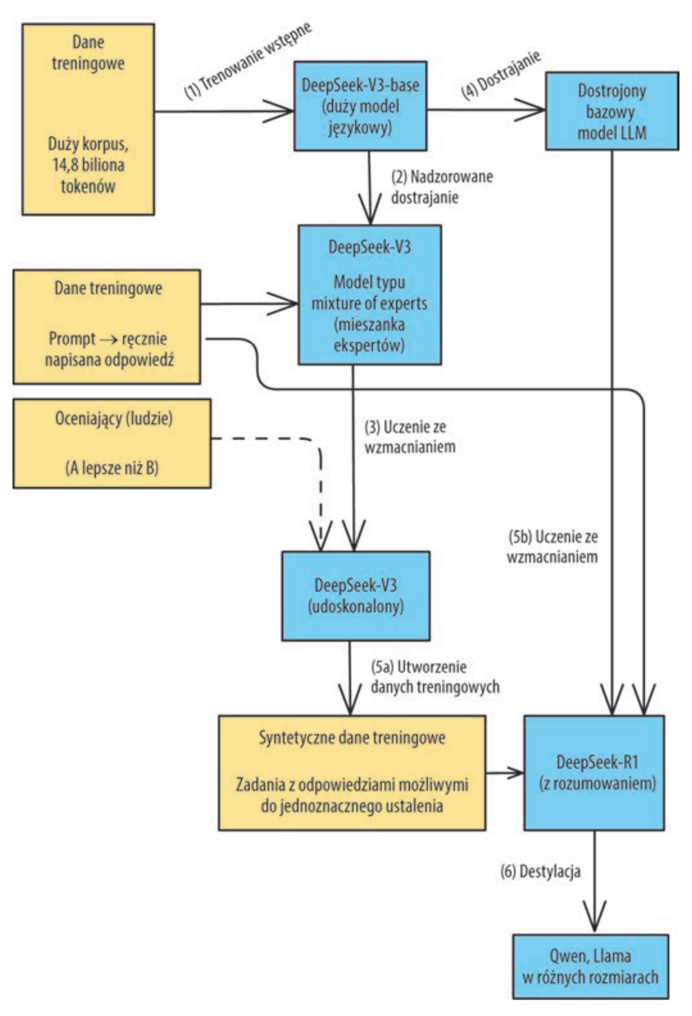
```
from pydantic_ai.models.openai import OpenAIModel
from pydantic_ai.providers.openai import OpenAIProvider

model = OpenAIModel(
    model_name='llama3.2',
    provider=OpenAIProvider(base_url='http://localhost:11434/v1')
)
```

Jak powstają modele bazowe?

W przeciwieństwie do tradycyjnych zastosowań uczenia maszynowego Twoje aplikacje AI rzadko obejmują etap treningu. Zamiast tego budujesz je na modelach bazowych ogólnego przeznaczenia, które wcześniej wytrenowano do wykonywania wielu typów zadań. W większości przypadków możesz pominąć wewnętrzne szczegóły działania modelu — ten punkt dodajemy wyłącznie po to, aby ułatwić zrozumienie używanego słownictwa.

Wiosną 2025 roku, gdy powstawała ta książka, DeepSeek był modelem bazowym, dla którego publicznie dostępnych było najwięcej informacji na temat procesu trenowania. Informacje te posłużą do omówienia kluczowych etapów tworzenia modelu bazowego (zobacz rysunek 1.2). Firmy OpenAI, Google, Anthropic i Meta przy tworzeniu modeli GPT, Gemini, Claude i Llama



Rysunek 1.2. Etapy treningu modelu DeepSeek-R1

mogły zastosować inne szczegółowe rozwiązania, jednak ogólny przebieg tych procesów jest najprawdopodobniej zbliżony.

Bazowy model językowy DeepSeek został wstępnie wytrenowany (krok 1. na rysunku 1.2) na zróżnicowanym, wysokiej jakości korpusie obejmującym 14,8 biliona tokenów. Dla porównania — wszystkie dzieła Szekspira to około 1,2 miliona tokenów, a więc zbiór treningowy DeepSeeka odpowiada mniej więcej „12 milionom Szekspirów”. W przeciwieństwie do wczesnych modeli LLM, które trenowano na słowach, współczesne modele są trenowane na *tokenach*, czyli krótkich sekwencjach znaków. Dzięki temu potrafią przyswajać elementy spoza słownika języka, na przykład nazwy własne. O wysokiej jakości modeli DeepSeek decyduje jednak nie tylko skala — zespół przypisuje ją również starannej kuracji danych, w tym rygorystycznym procesom deduplikacji (zob. *https://oreil.ly/5hjPn*).

Etap *trenowania wstępnego* (krok 1. na rysunku 1.2) polega na uczeniu modelu na ogromnym zbiorze tokenów w celu wykształcenia ogólnych zdolności rozumienia języka. Na tym etapie kluczowym zadaniem jest nauczenie modelu przewidywania kolejnego tokena na podstawie kontekstu tworzonego przez wcześniejsze tokeny w danych treningowych. Z tego powodu o dużych modelach językowych często mówi się, że są „predyktorami następnego tokena", jednak przewidywanie kolejnego tokena stanowi jedynie pierwszy etap całego procesu trenowania.

Po etapie trenowania wstępnego model przechodzi *nadzorowane dostrajanie* (ang. *supervised fine-tuning* — SFT; krok 2. na rysunku 1.2), którego celem jest poprawa zdolności wykonywania instrukcji oraz podniesienie jakości generowanych odpowiedzi. Na tym etapie wykorzystuje się starannie przygotowane zbiory przykładów pisanych przez ludzi. Firma Cohere informowała na przykład (*https://oreil.ly/qQhZu*), że do ulepszania swoich modeli angażuje lekarzy, analityków finansowych i księgowych. Można założyć, że ci specjaliści przygotowują wzorcowe odpowiedzi dla zadanych promptów. Rezultatem tego procesu jest DeepSeek-V3, czyli model typu *mixture of experts* (MoE) — rozwiązanie optymalizacyjne, które umożliwia zastosowanie bardzo dużej liczby parametrów przy jednoczesnej aktywacji jedynie ich części dla każdego tokena. DeepSeek-V3 ma łącznie 671 miliardów parametrów, jednak dla pojedynczego tokena aktywowanych jest tylko 37 miliardów (*https://arxiv.org/abs/2412.19437*). Takie podejście pozwala modelowi korzystać z różnych ścieżek obliczeniowych w zależności od rodzaju instrukcji.

Etap *uczenia ze wzmacnianiem* (krok 3. na rysunku 1.2) służy dalszemu dopracowaniu modelu na podstawie preferencji wyrażanych przez człowieka. Na tym etapie stosuje się technikę RLHF (ang. *reinforcement learning with human feedback* — dosłownie: uczenie ze wzmacnianiem z komentarzami człowieka), która polega na prezentowaniu oceniającym par wygenerowanych odpowiedzi i proszeniu ich o wskazanie tej, która lepiej spełnia oczekiwania. Takie dostrajanie preferencji pozwala lepiej dopasować odpowiedzi modelu do oczekiwań i wartości człowieka. Etap ten bywa również określany jako *optymalizacja preferencji*.

O ile DeepSeek-V3 powstał przy użyciu niewielkiej liczby przykładów pisanych przez ludzi oraz danych preferencyjnych, pełny model DeepSeek-R1 opracowano w wieloetapowym procesie (*https://oreil.ly/djxoR*) obejmującym:

1. *Zimny start* — dostrojenie modelu bazowego (DeepSeek-V3-Base) na tysiącach punktów danych, aby zbudować fundament.
2. *Czyste uczenie ze wzmacnianiem* (ang. *pure reinforcement learning*) — zastosowanie podejścia opartego wyłącznie na uczeniu ze wzmacnianiem w celu poprawy zdolności rozumowania.
3. *Próbkowanie odrzucające* (ang. *rejection sampling*) — wybór najlepszych przykładów z rozkładu danych powstałego podczas ostatniego udanego przebiegu RL.
4. *Nadzorowane dostrajanie* (SFT) — połączenie danych syntetycznych z danymi nadzorowanymi z DeepSeekV3-Base w takich dziedzinach jak pisanie, pytania i odpowiedzi faktograficzne (QA) oraz samoświadomość.
5. *Końcowe uczenie ze wzmacnianiem* — ostatni etap uczenia ze wzmacnianiem na zróżnicowanych promptach i z wykorzystaniem różnych scenariuszy.

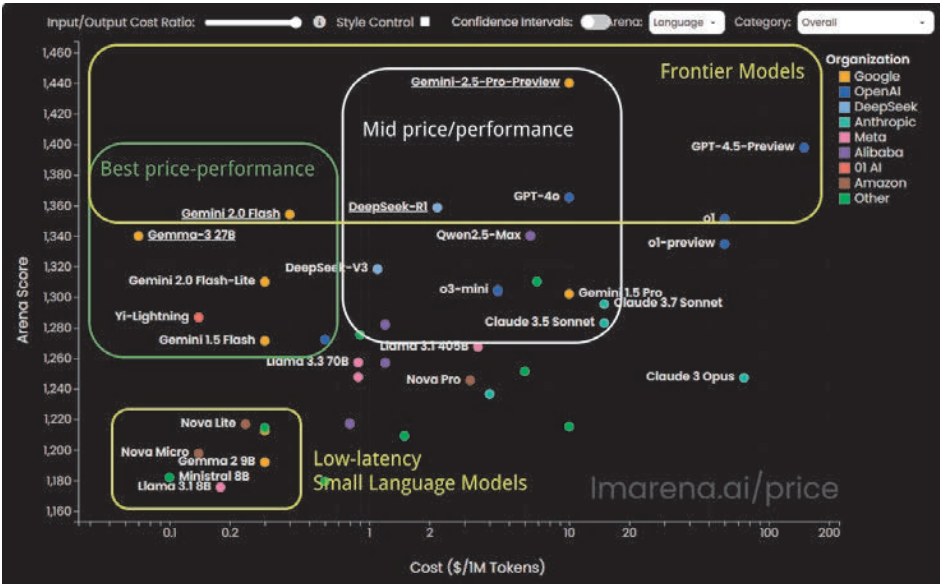
Przełomowy okazał się drugi krok (czyste uczenie ze wzmacnianiem), w którym zespół zastosował uczenie ze wzmacnianiem bezpośrednio na modelu bazowym (DeepSeek-V3-Base), *bez* korzystania we wstępnej fazie z techniki SFT. Dzięki temu przy rozwiązywaniu złożonych problemów model mógł eksplorować rozumowanie typu *chain-of-thought* (CoT), co doprowadziło do powstania modelu DeepSeek-R1-Zero (*https://oreil.ly/lCLIZ*). Okazało się, że zdolności rozumowania w LLM da się dostroić dzięki użyciu samej tylko techniki RL, bez SFT, choć wcześniej wielu badaczy uznawało SFT za konieczność. Ponieważ przykłady pisane przez ludzi są kosztowne, czyste RL pozwala trenować model dłużej i na znacznie bardziej zróżnicowanym zbiorze problemów.

Aby zwiększyć dostępność możliwości DeepSeek-R1, zespół przygotował wersje destylowane modelu (*https://oreil.ly/9G-r8*), które można uruchamiać na skromniejszym sprzęcie, przy zachowaniu dużej części pierwotnych zdolności rozumowania. Obejmują one modele oparte na architekturach Qwen i Llama, dostępne w różnych rozmiarach (1,5B, 7B, 8B, 14B itd.).

Krajobraz modeli bazowych

Ekosystem modeli bazowych GenAI szybko się rozwinął. W efekcie powstało kilka kategorii odpowiadających różnym potrzebom.

Ponieważ akademickie benchmarki nie pozwalają już miarodajnie porównywać modeli i są podatne na manipulację, obecnie najpowszechniej akceptowanym sposobem ich oceny pozostaje porównywanie modeli parami w ślepych testach. Takie porównania na dużą skalę prowadzi LMArena (*https://lmarena.ai*), a tabelę wyników z kwietnia 2025 roku pokazano na rysunku 1.3.



Rysunek 1.3. Ranking LMArena z 6 kwietnia 2025 r. (ramki dodali autorzy)

Na wykresie na osi *Y* zaprezentowano ocenę modelu, natomiast na osi *X* — koszty. Dzięki temu możesz ustalić, który model najlepiej mieści się w Twoim budżecie oraz jakiej minimalnej ceny należy się spodziewać dla danej oceny. Zwróć uwagę, że skala Elo na osi *Y* przedstawia zależność logarytmiczną[2] — podobnie jak koszt na osi *X*.

W dniu, w którym autorzy wykonali zrzut ekranu, najwyżej oceniane modele (zgodnie z osią *Y*) to Gemini 2.5 Pro Preview, GPT-4.5 (Preview), GPT-4o oraz DeepSeek-R1. Niewiele niżej plasowały się Gemini 2.0 Flash i OpenAI o1. Ranking zmienia się codziennie, gdy pojawiają się nowe modele i przybywa ocen, ale w dłuższym okresie pozostaje zaskakująco stabilny — flagowe modele Google, OpenAI i Anthropic niemal zawsze są na szczycie. Te modele określa się zbiorczo jako *modele wiodące* (ang. *frontier models*).

Modele wiodące, takie jak GPT-5 i Gemini 2.5 Pro, reprezentują aktualny stan zaawansowania w zakresie możliwości modeli językowych i oferują najwyższą wydajność w zadaniach wymagających rozumowania, wiedzy oraz przetwarzania multimodalnego. Są jednak zasobożerne i kosztowne, a ze względu na rozmiar oraz własnościowy charakter nie można ich uruchamiać lokalnie. Stosuje się je w aplikacjach klasy enterprise, które wymagają zaawansowanego rozumowania i w których szybkość działania nie stanowi kluczowego ograniczenia. Do najnowszych kierunków rozwoju modeli wiodących należą obsługa multimodalności, wzmocnione mechanizmy rozumowania oraz znacznie rozszerzone okna kontekstu, sięgające w niektórych przypadkach nawet 2 milionów tokenów.

Wersje destylowane modeli wiodących zapewniają kompromis między jakością a efektywnością i oferują sensowne możliwości przy niższych kosztach oraz krótszym czasie odpowiedzi. Do czołowych przykładów należą Gemini Flash, Claude Sonnet oraz GPT-4o-mini. Modele te zwykle dobrze radzą sobie z typowymi zadaniami, takimi jak generowanie treści i streszczanie. Zapewniają także szybki czas reakcji i opłacalność w zastosowaniach o dużej skali. Różnica kosztów między Gemini Pro a Gemini Flash staje się szczególnie wyraźna po uwzględnieniu faktu, że oś *X* na rysunku 1.3 ma skalę logarytmiczną — Flash jest dwudziestokrotnie tańszy.

Modele *o otwartych wagach* (ang. *open-weight*) udostępniają parametry publicznie, co daje przejrzystość, umożliwia rozwój społecznościowy i pozwala na dostrajanie. Przykłady to Llama, Mistral, DeepSeek, Qwen i Falcon. Te modele pozwalają osiągać solidne wyniki, choć zwykle ustępują modelom wiodącym. Można je dostrajać na danych zastrzeżonych, ale hosting wymaga większej wiedzy. Na rynku istnieją jednak usługi API, takie jak Together.ai, oraz w pełni zarządzane punkty końcowe czołowych dostawców chmurowych, które ograniczają ten problem.

Modele, które można hostować lokalnie, projektuje się z myślą o uruchamianiu na sprzęcie konsumenckim lub korporacyjnym bez konieczności nawiązywania połączenia z chmurą. Przykładami są Llama 8B i Gemma 2B, których sprzętowo zoptymalizowane wersje są dostępne w ramach NVIDIA NIM (*https://developer.nvidia.com/nim*). Takie podejście zapewnia pełną prywatność — dane nie opuszczają urządzeń lokalnych — oraz eliminuje bieżące koszty wywołań API. Ceną jest jednak wyraźnie mniejszy zakres możliwości w porównaniu z modelami

---

[2] Różnica 400 punktów w rankingu Elo odpowiada stosunkowi szans 10 : 1 na zwycięstwo gracza o wyższej ocenie nad graczem o niższej ocenie.

chmurowymi. Zapotrzebowanie na modele wiodące w systemach typu air-gapped, czyli odciętych od internetu, skłoniło część dostawców zastrzeżonych modeli do oferowania także takich rozwiązań. Przykładowo Gemini można uruchamiać lokalnie w środowisku Google Distributed Cloud (*https://oreil.ly/HFIut*), a modele OpenAI można wdrażać w lokalnej infrastrukturze opartej na platformie Azure (*https://oreil.ly/-XSfk*).

Agentowa AI

W przykładzie z `PydanticAI` klasa wywołująca model Claude nazywała się `Agent`:

```
from pydantic_ai import Agent
agent = Agent('anthropic:claude-3-7-sonnet-latest', ...)
```

Czym jest agent? W informatyce termin „agent” od dawna opisuje byt programowy, który działa w imieniu użytkownika lub innego programu. Gdy wywołujesz model bazowy, określasz rolę, podajesz kontekst i prosisz o wykonanie instrukcji. W tym znaczeniu LLM działa jako Twój agent.

Oto przykład utworzenia agenta, który ma zarządzać stanami magazynowymi w sklepie (pełny kod jest na GitHubie — *https://oreil.ly/d2kyK*):

```
agent = Agent(
    f"anthropic:{MODEL_ID}",
    system_prompt="You are an inventory manager who orders just in time.",
    ...
)
```

Autonomia

W AI oczekuje się również, że agent będzie w pewnym stopniu autonomiczny. W tym ujęciu LLM pełni funkcję „mózgu” agenta, więc nie musisz opisywać krok po kroku, jak ma zarządzać magazynem — wystarczy *cel*, czyli zamawianie „dokładnie na czas”.

Załóżmy, że masz listę pozycji magazynowych oraz dane o sprzedaży i czasie dostawy:

```
@dataclass
class InventoryItem:
    name: str
    quantity_on_hand: int
    weekly_quantity_sold_past_n_weeks: [int]
    weeks_to_deliver: int

items = [
    InventoryItem("itemA", 300, [50, 70, 80, 100], 2),
    InventoryItem("itemB", 100, [70, 80, 90, 70], 2),
    InventoryItem("itemC", 200, [80, 70, 90, 80], 1)
]
```

Przekaż listę agentowi, a agent sam ustali, jakie zamówienie trzeba złożyć:

```
result = agent.run_sync(f"""
Identify which of these items need to be reordered this week.
```

```
**Items**
{items}
""")
```

Wynik będzie zawierał między innymi:

> itemB
>
> **quantity_to_order=300** reason_to_reorder='Current stock (100) is insufficient to cover projected demand over delivery time. Based on recent weekly sales (70-90 units), we need to **order enough to cover the 2-week delivery period** plus maintain safety stock.'

Warto porównać powyższy sposób z tradycyjnym programowaniem, gdzie musiałbyś jawnie zaimplementować logikę zarządzania zapasami. Właśnie *autonomia* — zdolność do działania bez ciągłego kierowania przez człowieka i bez programowania każdego kroku — stanowi kluczową różnicę między klasycznym oprogramowaniem a agentami AI.

Cechy agentów

Poza autonomią od agentów zwykle oczekuje się także następujących cech:

Orientacja na cel

Agent nie tylko reaguje na prompty, ale przede wszystkim dąży do uzyskania określonych rezultatów. Cel agenta zarządzającego magazynem — zamawianie „dokładnie na czas" — został określony w prompcie systemowym.

Planowanie i wnioskowanie

Zauważ, że agent potrafił zaplanować kroki potrzebne do ustalenia liczby sztuk do zamówienia. Określił zakres ostatniej tygodniowej sprzedaży, przeniósł maksymalny poziom sprzedaży na okres dostawy, policzył, ile sztuk będzie potrzebnych, a następnie wyliczył liczbę sztuk do zamówienia. Nie trzeba było tego programować wprost.

Pozyskiwanie informacji i działanie

Agent potrafi zbierać potrzebne informacje z otoczenia oraz wykonywać działania w systemach zewnętrznych. Zazwyczaj umożliwia się to przez pozwolenie mu na wywoływanie funkcji zewnętrznych, takich jak wyszukiwanie w sieci, korzystanie z kalkulatorów czy zapisywanie danych w bazach, za pomocą mechanizmu Tool Calling (dosłownie: wywoływanie narzędzi — wzorzec 21.), który omawiamy w rozdziale 7. Dzięki Tool Calling nie trzeba jawnie przekazywać agentowi listy pozycji dla menedżera magazynu. Agent może samodzielnie pobierać z systemów backendowych informacje o stanie zapasów i tygodniowej sprzedaży. Co więcej, zamiast jedynie informować, że pozycję B należy zamówić, może wywołać odpowiednie API — nawet API po stronie dostawcy — i złożyć zamówienie.

Adaptacja i uczenie się

Skąd wiadomo, że zamówienie 300 sztuk jest właściwe? Kierownik magazynu przyjmuje założenia dotyczące sprzedaży w kolejnych dwóch tygodniach i sprawdza, czy sklep nie odnotuje braków. Agent może przeprowadzić dokładnie taką samą weryfikację. W rozdziałach 6. i 9. pokazujemy takie wzorce, jak Reflection (wzorzec 18.) oraz Self-Check (wzorzec 31.), które umożliwiają agentowi ocenę własnych wyników i ich korektę.

W trakcie pisania tej książki zachowania agentowe pozostają w dużej mierze celem aspiracyjnym dla aplikacji opartych na modelach bazowych, ponieważ brak deterministycznego działania, halucynacje oraz inne tryby awarii utrudniają budowę w pełni autonomicznych systemów AI. Dobrym przykładem jest brak powtarzalności wyników: przy każdym uruchomieniu system może wygenerować inną listę pozycji do zamówienia oraz inne ilości. Planowanie sprawdza się w prostych scenariuszach, lecz zawodzi w bardziej złożonych. W rozdziale 5. przedstawiamy takie wzorce jak Chain of Thought (wzorzec 13.), które pozwalają poprawiać zdolności agenta do planowania i wnioskowania. Wiele wzorców opisanych w tej książce stanowi próbę nadania aplikacjom AI bardziej agentowego charakteru albo przynajmniej przesunięcia granic tego, co można obecnie zbudować.

Precyzyjna kontrola

Modele bazowe dają możliwość precyzyjnego sterowania procesem generowania dzięki kontroli próbkowania oraz mechanizmu przeszukiwania wiązkowego (ang. *beam search*), które omawiamy w dalszej części tego rozdziału. Ustawienia generowania w modelach LLM pozwalają ustalić kompromis pomiędzy przewidywalnością i jakością odpowiedzi a ich kreatywnością i różnorodnością. Zrozumienie matematycznych podstaw tych ustawień, od logitów po strategie próbkowania, daje bezpośredni wgląd w sposób działania modelu i często okazuje się skuteczniejsze niż stosowanie wzorców projektowych omawianych w kolejnych rozdziałach książki.

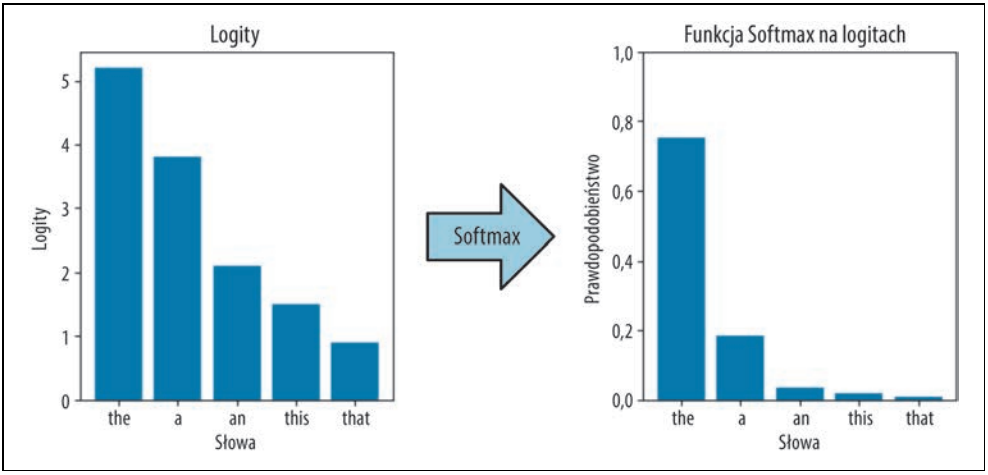
Logity

Modele językowe składają się z setek warstw, ale ostatnia warstwa przewiduje następne słowo, aby kontynuować generowanie tekstu. Model nie prognozuje jednego tokena. Zamiast tego podaje zestaw kandydatów i prawdopodobieństwo, że każdy z nich będzie następnym tokenem.

Logity to surowe, nienormalizowane wartości z ostatniej warstwy modelu, zanim zostaną przekształcone w prawdopodobieństwa. Odzwierciedlają one ocenę modelu dotyczącą tego, jak bardzo prawdopodobne jest, że dany token ze słownika stanie się kolejnym elementem sekwencji. Dla uproszczenia przyjmijmy, że istnieje pięć możliwych kontynuacji generowanej sekwencji, a odpowiadające im logity przedstawiono po lewej stronie rysunku 1.4.

Gdyby model stosował próbkowanie zachłanne (ang. *greedy sampling*), wybierałby wyłącznie najbardziej prawdopodobne słowo, czyli w języku angielskim słowo *the*, bo ma najwyższą wartość logitu. Taka strategia prowadzi jednak do powtarzalnego i mało interesującego tekstu, więc modele stosują próbkowanie, w którym wszystkie słowa-kontynuacje mają niezerową szansę wyboru. Przekształcenie logitów w prawdopodobieństwa odbywa się przez funkcję softmax:

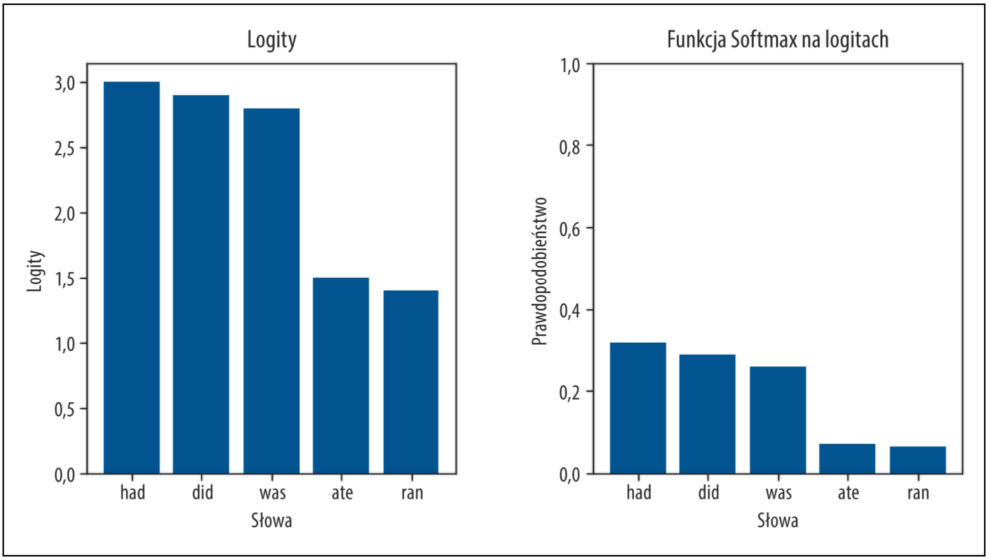
$$P(token_i) = e^{logit_i} / \sum_j e^{logit_j}$$



Rysunek 1.4. Wykonanie funkcji softmax na logitach pozwala uzyskać prawdopodobieństwa różnych kontynuacji

W tym zapisie `P(token_i)` oznacza prawdopodobieństwo wyboru `tokena_i`. Funkcja `softmax` wzmacnia „szczyty” rozkładu i tłumi „ogon”. Widać to na rysunku 1.4, jeśli porównasz słupki dla *the* i *a* przed zastosowaniem i po zastosowaniu funkcji `softmax`.

Jeśli rozkład możliwych kontynuacji jest bardziej wyrównany, jak na rysunku 1.5, wpływ funkcji `softmax` jest słabszy.



Rysunek 1.5. Gdy rozkład jest bardziej wyrównany, wpływ funkcji softmax jest mniej wyraźny

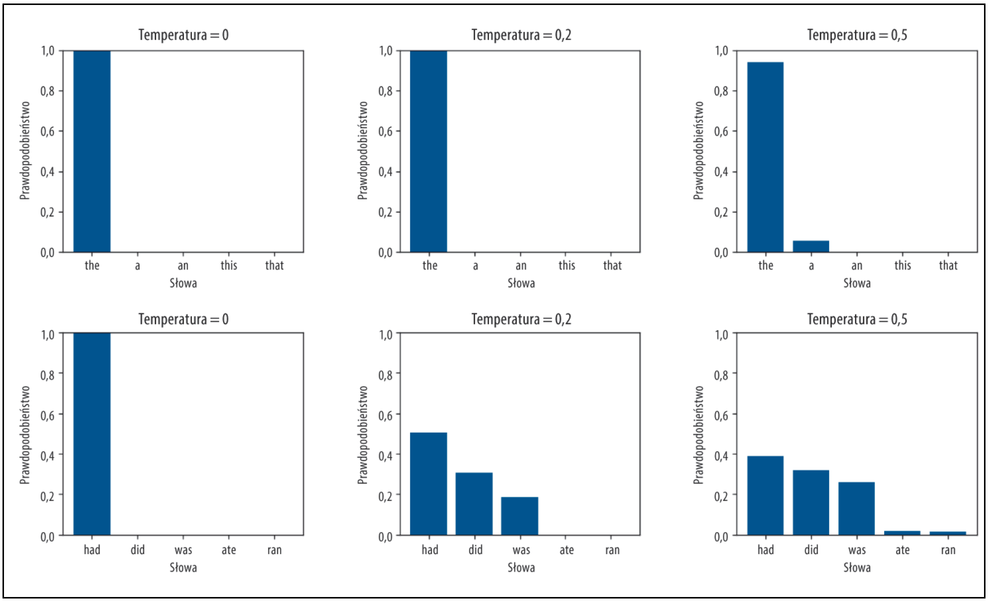
Sposób korzystania z logitów do zarządzania stylem zaprezentujemy w rozdziale 2.

Temperatura

Temperatura (*T*) to hiperparametr, który steruje losowością wyboru tokenów. Działa przez skalowanie logitów przed przepuszczeniem ich przez softmax. Zmienione równanie wygląda tak:

$$P(token_i) = e^{logit_i/T} \;/\; \sum_j e^{logit_j/T}$$

Wpływ skalowania tych samych wykresów przy różnych wartościach *T* przedstawiono na rysunku 1.6.



Rysunek 1.6. Wpływ temperatury na skalowanie logitów

Jak widać, ustawienie temperatury na zero włącza próbkowanie zachłanne. Gdy temperatura rośnie, rośnie też szansa wyboru słów z „ogona”, czyli mniej prawdopodobnych. Efekt temperatury jest słabszy, gdy rozkład od początku jest bardziej wyrównany.

W bibliotece `PydanticAI` temperaturę ustawia się w następujący sposób:

```
agent = Agent(
    'anthropic:claude-3-7-sonnet-latest',
    model_settings={
        "temperature": 0.5
    },
    system_prompt="Dokończ zdanie."
)
```

Z kolei w przypadku bezpośredniego korzystania z API Anthropic robimy to jak poniżej:

```
completion = client.messages.create(
    model="claude-3-7-sonnet-latest",
    system="Complete the sentence.",
```

```
        temperature=0.5,
        messages=[
            ...
        ]
    )
```

Przykład pokazuje różne kontynuacje frazy *The trade war caused* (*Wojna handlowa spowodowała*), uzyskane dla trzech ustawień temperatury:

> 0.0: The trade war caused significant disruptions to global supply chains, leading to increased prices for consumers and economic uncertainty for businesses across multiple industries.
>
> 0.5: The trade war caused significant disruptions to global supply chains, leading to increased prices for consumers and economic uncertainty for businesses across multiple industries. Many manufacturers were forced to reconsider their production strategies, while farmers faced reduced export opportunities as retaliatory tariffs limited access to international markets. The long-term effects included accelerated efforts to diversify supply chains away from affected regions and renewed debates about the effectiveness of protectionist trade policies.
>
> 0.8: The trade war caused significant disruptions to global supply chains, forcing many companies to reconsider their manufacturing strategies and sourcing policies. It led to increased tariffs on imported goods, higher prices for consumers, and economic uncertainty in affected industries. Several businesses reported decreased profits as they absorbed additional costs or lost market share in foreign markets. The prolonged tension also contributed to volatility in financial markets and complicated diplomatic relations between the involved nations.

Ustawienie wyższej temperatury zwykle daje bardziej kreatywny wynik.

W rozdziałach 3. i 6. opisano zastosowanie takich technik, jak RAG oraz LLM-as-Judge, przy których pożądane jest korzystanie z niskiej temperatury. W niektórych sytuacjach trzeba nawet ustawić temperaturę zerową.

Próbkowanie top-K

Próbkowanie top-K ogranicza wybór tokena do *k* najbardziej prawdopodobnych tokenów ze słownika, czyli obcina długi „ogon” rozkładu prawdopodobieństwa. To ogranicza ryzyko generowania przypadkowych lub nielogicznych kontynuacji przy wysokiej temperaturze.

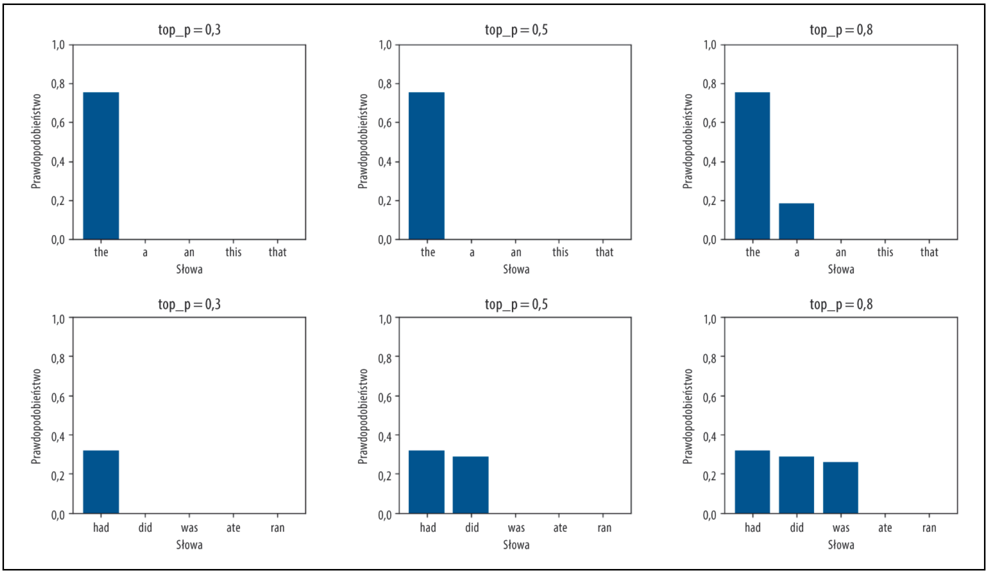
Oto wpływ różnych wartości top-K na kontynuację frazy *The spaceship* (*Statek kosmiczny*):

> 1: The spaceship zoomed through the vast expanse of space, its powerful engines glowing blue against the darkness as it carried its crew toward distant stars and unknown adventures.
>
> 10: The spaceship glided silently through the vast emptiness of space, its powerful engines propelling it toward the distant galaxy where no human had ventured before.
>
> 100: The spaceship soared through the starry expanse, its gleaming hull reflecting the distant light of alien suns as it carried its crew toward unexplored worlds beyond the edge of known space.

Jak można zauważyć, gdy wartość top-K jest niska, generowany tekst zwykle jest bardzo podobny do sformułowań znanych z klasycznej fantastyki.

Próbkowanie jądra

Próbkowanie jądra polega na dynamicznym wyborze najmniejszego zbioru tokenów, którego łączne prawdopodobieństwo przekracza próg *p*. Z tego powodu metoda ta bywa również określana jako *próbkowanie top-P*. W ten sposób powstaje „jądro” tokenów obejmujące większość rozkładu prawdopodobieństwa. Wpływ zastosowania różnych wartości parametru top-P na przykładowe rozkłady zilustrowano na rysunku 1.7.



Rysunek 1.7. Wpływ próbkowania top-P

Próbkowanie jądra w każdym kroku dostosowuje się do stopnia pewności modelu. Gdy model działa z dużą pewnością, bierze pod uwagę niewielką liczbę tokenów, podobnie jak przy niskiej wartości top-K. Gdy poziom pewności jest niższy, uwzględnia większą liczbę tokenów, jak ma to miejsce przy wysokim top-K. W rezultacie próbkowanie top-P zazwyczaj prowadzi do bardziej naturalnego tekstu niż stosowanie stałej wartości top-K.

Wyszukiwanie wiązkowe

Modele nie wybierają tokenów kontynuacji pojedynczo. Stosują *wyszukiwanie wiązkowe* (ang. *beam search*), czyli deterministyczną metodę wyszukiwania, której celem jest znalezienie najbardziej prawdopodobnej sekwencji tokenów przez równoległe rozpatrywanie wielu możliwych kontynuacji. Dzięki temu model bierze pod uwagę prawdopodobieństwo całej sekwencji, a nie wyłącznie kolejnego tokena.

Część modeli udostępnia opcje do sterowania mechanizmem wyszukiwania wiązkowego.

Mechanizmy *ograniczenia powtórzeń* (ang. *repetition penalties*) obniżają prawdopodobieństwo ponownego pojawienia się tokenów, które już wystąpiły w wygenerowanym tekście, co ogranicza ryzyko zapętlenia. Ograniczenia te obejmują między innymi:

Ograniczenie częstości (ang. *frequency penalty*)

Jego wpływ rośnie wraz z liczbą wystąpień danego słowa lub frazy w wygenerowanym tekście.

Ograniczenie obecności (ang. *presence penalty*)

Jest stosowane przy każdym powtórzeniu i sprzyja większej różnorodności używanych tokenów.

Ograniczenia długości (ang. *length penalties*) modyfikują ocenę sekwencji w zależności od długości. Obejmują one następujące elementy:

Minimalna długość (ang. *minimum length penalty*)

Wymusza wygenerowanie co najmniej określonej liczby tokenów.

Maksymalna długość (ang. *maximum length penalty*)

Ogranicza generację do określonej długości.

Normalizacja długości (ang. *length normalization penalty*).

Dzieli wyniki sekwencji przez funkcję długości, aby ograniczyć stronniczość w kierunku krótszych sekwencji.

Szerokość wiązki wyszukiwania (ang. *beam search width*)

To liczba równoległych sekwencji uwzględnianych podczas generowania. Przykład użycia tego parametru zaprezentowano w opisach wzorców w rozdziałach 2. i 6.

W chwili pisania książki OpenAI i Gemini obsługiwały mechanizmy ograniczania powtórzeń, natomiast Anthropic ich nie udostępniało. Biblioteka Transformers na platformie Hugging Face oferowała ograniczenia długości sekwencji oraz możliwość sterowania szerokością wiązki wyszukiwania, jednak modele dostępne w formie usług hostowanych zazwyczaj tych opcji nie zapewniały.

Uczenie w kontekście

W „tradycyjnym” uczeniu maszynowym, gdzie modele trenuje się głównie pod kątem wykonywania jednego zadania (np. klasyfikacji lub regresji), dopasowanie do nowego zadania wymaga treningu wag lub parametrów. W przypadku modeli LLM wystarczy zmienić prompt i pozostawić wagi bez zmian, aby można je było wykorzystać do realizacji bardzo różnych zadań.

Zdolność modeli bazowych do adaptacji do nowych zadań wyłącznie na podstawie przykładów lub instrukcji w prompcie nazywa się *uczeniem w kontekście* (ang. *in-context learning*). Wynika ona z treningu na zróżnicowanych zbiorach danych, zawierających wiele formatów zadań i instrukcji. Gdy prompt zawiera przykłady, model potrafi rozpoznać wzorzec i zastosować go do nowych przypadków. Wagi modelu się nie zmieniają — do wykonania nowych zadań model wykorzystuje swoją dotychczasową wiedzę.

Uczenie typu zero-shot

Uczenie typu zero-shot (ang. *zero-shot learning*) polega na przekazaniu modelowi polecenia do wykonania zadania bez podawania przykładów. Technika opiera się na wiedzy z treningu wstępnego i na zdolności rozumienia instrukcji w języku naturalnym.

Na przykład możesz poprosić Gemini-Flash:

> Przeanalizuj użycie światła w obrazie Claude'a Moneta *Impresja, wschód słońca* i wyjaśnij, w jaki sposób dzieło pokazuje techniki impresjonistyczne.

Odpowiedź mogłaby wyglądać mniej więcej tak (choć wynik może się różnić, ponieważ odpowiedzi modelu są niedeterministyczne):

> W obrazie *Impresja, wschód słońca* Monet wykorzystuje światło do uchwycenia ulotnej chwili, przedkładając jego przemijające efekty ponad precyzyjne detale. Zamglona atmosfera oraz intensywne refleksy na wodzie podkreślają impresjonistyczne skupienie na percepcji i subiektywnym doświadczaniu światła, a nie na realistycznym odwzorowaniu sceny. To dążenie do uchwycenia bezpośredniego „wrażenia" światła stało się jednym z wyróżników ruchu impresjonistycznego.

Uczenie typu few-shot

Uczenie typu few-shot (ang. *few-shot learning*) jest rozszerzeniem podejścia uczenia w kontekście — w prompcie umieszczasz kilka przykładów. Przykłady pokazują strukturę zadania i oczekiwany format odpowiedzi, a model przenosi tę strukturę na nowe przypadki. Ponieważ przykłady znajdują się w kontekście promptu, jest to także forma *uczenia w kontekście*.

Uczenie typu few-shot to prosta i skuteczna forma *inżynierii kontekstu* — polega na dodawaniu do kontekstu promptu starannie przygotowanych danych i instrukcji w celu uzyskania pożądanej jakości. W tej książce znajdziesz wiele wzorców opartych na koncepcji uczenia typu few-shot (zwłaszcza w rozdziale 5.).

Załóżmy, że chcesz zbudować aplikację, która pozwala użytkownikowi opisać obraz słowami, a następnie rozpoznaje dzieło. Możesz pokazać modelowi, jak ma wyglądać identyfikacja, podając przykład pytania i odpowiedzi dotyczących obrazu Claude'a Moneta *Impresja, wschód słońca*:

~~~
agent = Agent(
    MODEL_ID,    system_prompt="""You are an expert on art history. I will describe
a painting. You should identify it.
"""
)

result = agent.run_sync("""
Example:
```
Description: shows two small rowboats in the foreground and a red Sun.
Answer:
    Painting: Impression, Sunrise
    Artist: Claude Monet
    Year: 1872
~~~

```
    Significance: Gave the Impressionist movement its name; captured the fleeting
    ↪effects of light and atmosphere, with loose brushstrokes.
```

Description: The painting shows a group of people eating at a table under an outside
↪tent. The men are wearing boating hats.
""")
```

Wynik poprawnie identyfikuje obraz Renoira (*https://oreil.ly/4d-5S*), który odpowiada podanemu opisowi:

> Answer:
>
> Painting: Luncheon of the Boating Party
>
> Artist: Pierre-Auguste Renoir
>
> Year: 1881
>
> Significance: Captures a joyful social gathering of Renoir's friends at the Maison Fournaise restaurant in Chatou, France, depicting the carefree atmosphere of Parisian life during the Belle Époque; known for its vibrant colors, lively composition, and portrayal of light and shadow.

Zamiast budować model wyspecjalizowany dla danej dziedziny, gromadzić obszerne zbiory danych treningowych i przeprowadzać proces treningu, *uczenie w kontekście* pozwala modelom LLM wykonywać nowe zadania „w locie". Przykłady zawarte w prompcie pełnią rolę wskazówek, które pomagają modelowi naśladować zaprezentowane wzorce relacji wejście – wyjście.

Gdy uczenie zero-shot — czyli bez podawania przykładów — nie rozwiązuje problemu, warto dodać do promptu przykłady i przekazać modelowi konkretne instrukcje wykonania zadania. Uczenie w kontekście z użyciem kilku przykładów jest prostsze niż stosowanie klasycznego uczenia maszynowego, ponieważ dostrajanie modelu nie wymaga przygotowywania rozbudowanego zbioru danych. W praktyce zastosowanie techniki *uczenia few-shot* często pozwala osiągać bardzo dobre rezultaty.

Uczenie w kontekście umożliwia szybkie prototypowanie rozwiązań i sprawdza się także w zastosowaniach produkcyjnych — aktualizacja przykładów w prompcie jest znacznie szybsza niż kuracja danych, ponowny trening i wdrożenie modelu ML.

Uczenie w kontekście ma jednak pewne ograniczenia, ponieważ:

- działa tylko wtedy, gdy model bazowy dysponuje już odpowiednią wiedzą i kompetencjami;
- dodawanie wielu przykładów powoduje zużywanie cennych tokenów okna kontekstu i spowalnia wnioskowanie;
- modele LLM miewają trudności z uogólnianiem bardziej złożonych problemów na podstawie niewielu przykładów.

W takich sytuacjach lepszym rozwiązaniem może być post-trening.
```

# Trening doskonalący

Metody *treningu doskonalącego* (ang. *post-training*) polegają na modyfikowaniu wag wcześniej wytrenowanych modeli w celu dostosowania ich do nowych zadań lub dziedzin. Model po post-treningu trzeba wdrożyć i wykorzystywać z innego punktu końcowego niż model bazowy, który stanowił punkt wyjścia.

## Metody treningu doskonalącego

Jak pokazano na rysunku 1.2, trening modelu bazowego przebiega etapami. W pierwszym etapie trenuje się podstawowy model LLM do przewidywania kolejnego słowa, w drugim etapie uczy się go wykonywania zadań za pomocą SFT, a w trzecim stosuje się RLHF. Post-trening można przeprowadzić za pomocą dowolnego z tych sposobów — albo kilku z nich jednocześnie.

Poniżej przedstawiono pojęcia, z którymi w kontekście post-treningu spotykamy się najczęściej. Nie są to metody całkowicie rozłączne — częściowo pokrywają się ze sobą.

*Kontynuacja treningu wstępnego* (ang. *continued pretraining* — CPT)

Jeśli dysponujesz zbiorem danych zawierającym słownictwo (na przykład żargon branżowy) oraz powiązania, których model bazowy nie poznał w pierwotnym treningu, możesz kontynuować wstępny trening modelu bazowego. Wymaga to dostępu do pełnych wag i architektury modelu bazowego. Warto podkreślić, że trenowanie modeli bazowych jest niezwykle kosztowne i czasochłonne, a dodatkowo trzeba potem przeprowadzić dla nowo powstałego modelu pozostałe etapy treningu. W marcu 2023 roku Bloomberg zastosował to podejście do dokumentów finansowych (*https://oreil.ly/1AeUi*), jednak już kilka miesięcy później się okazało, że modele bazowe wyraźnie przewyższają (*https://arxiv.org/abs/2305.05862*) model wyspecjalizowany dziedzinowo. Od tego czasu niewiele organizacji decyduje się na tę strategię.

*Nadzorowane dostrajanie* (ang. *supervised fine-tuning* — SFT)

Umożliwia dalsze trenowanie modeli językowych w trybie nadzorowanym na zbiorach danych złożonych z par (prompt, odpowiedź). Wykorzystanie promptów zawierających konkretne instrukcje, takie jak „ulepsz ten plan zarządzania”, w połączeniu z szerokim i zróżnicowanym materiałem treningowym poprawia zdolność modelu do wykonywania tego typu poleceń formułowanych w języku naturalnym. Trenowanie na bogatym zestawie instrukcji wzmacnia także zdolności zero-shot w wielu różnych zadaniach. Takie *dostrajanie instrukcyjne* może również poprawić zdolność modelu do uogólniania oraz zwiększyć użyteczność i rzetelność jego odpowiedzi. Ostateczny efekt zależy jednak od wielkości i różnorodności zbioru danych: zastosowanie SFT wyłącznie do jednego zadania lub jednej instrukcji zwykle prowadzi do zapominania wcześniej nabytych umiejętności i braku uogólniania na nowe zadania. Jeżeli natomiast zbiór treningowy obejmuje wiele zadań, model może uogólniać na nowe zadania nawet po jednej rundzie SFT.

*Dostrajanie efektywne parametrowo* (ang. *parameter-efficient fine-tuning* — PeFT)

Ze względu na ogromne rozmiary modeli bazowych ich dalszy trening bywa niepraktyczny. Z tego powodu powstały metody *dostrajania efektywnego parametrowo*, które upraszczają proces treningu dużych modeli. Jedną z najpopularniejszych metod jest *adaptacja niskiego rzędu* (ang. *low-rank adaptation* — LoRA), w której dzięki dekompozycji niskiego rzędu aktualizacje wag reprezentuje się za pomocą mniejszych macierzy. Zamiast dostrajać wszystkie parametry modelu, LoRA zamraża oryginalne, wstępnie wytrenowane wagi i dodaje niewielkie, trenowalne macierze „adapterów", które podlegają dalszej dekompozycji na reprezentacje niskiego rzędu. LoRA radykalnie zmniejsza liczbę trenowalnych parametrów — nawet do 10 000 razy — oraz ogranicza zapotrzebowanie na pamięć GPU — nawet trzykrotnie. Nie zwiększa przy tym opóźnień podczas wnioskowania i często pozwala osiągać wyniki porównywalne z pełnym dostrajaniem. Rozszerzeniem metody LoRA jest QLoRA (ang. *quantization-aware low-rank adaptation* — dosłownie: adaptacja niskiego rzędu z uwzględnieniem kwantyzacji), w której wszystkie wagi modelu podlegają kwantyzacji. Proces treningu jest w tym przypadku bardziej oszczędny pamięciowo, choć wolniejszy. Z kolei model po dostrojeniu i kwantyzacji zajmuje mniej miejsca, a dzięki temu działa szybciej niż w wariancie LoRA.

*Strojenie preferencji* (ang. *preference tuning*)

Model można poddać post-treningowi poprzez wygenerowanie dwóch odpowiedzi na ten sam prompt i przekazanie informacji zwrotnej, która z nich jest lepsza. Gdy takie preferencje pochodzą od ludzi, a sam post-trening odbywa się z użyciem uczenia ze wzmacnianiem, mamy do czynienia z metodą RLHF. Często jednak stosuje się metodę DPO (ang. *direct preference optimization* — dosłownie: bezpośrednia optymalizacja preferencji), ponieważ jest bardziej efektywna obliczeniowo. DeepSeek wprowadził metodę GRPO (ang. *group relative policy optimization* — dosłownie: optymalizacja polityki względem grup), w której generuje się wiele odpowiedzi, a każdej z nich przypisuje się wynik znormalizowany względem średniej nagrody w danej grupie.

Rodzaj treningu doskonalącego, który możesz zastosować, jest ściśle związany ze strukturą zbioru danych. Jeżeli zbiór zawiera wyłącznie uzupełnienia tekstu, można go wykorzystać do treningu nienadzorowanego, na przykład do uczenia modelu nowego słownictwa lub — dzięki metodzie CPT — zupełnie nowych powiązań. Jeżeli zbiór składa się z idealnych odpowiedzi na dane wejściowe, czyli par wejście – wyjście, nadaje się do metod SFT lub dostrajania instrukcyjnego. Jeżeli natomiast zbiór zawiera po dwie odpowiedzi dla każdego wejścia wraz z informacją, która z nich jest lepsza, można zastosować strojenie preferencji. Każdą z tych form post-treningu da się przeprowadzić w sposób efektywny parametrowo lub na modelach kwantyzowanych.

Z treningiem doskonalącym zetkniesz się w kilku wzorcach opisanych w tej książce, między innymi w Content Optimization (wzorzec 5. w rozdziale 2.), Adapter Tuning (wzorzec 15. W rozdziale 5.) oraz Prompt Optimization (wzorzec 20. w rozdziale 6.). W chwili pisania tej książki (czerwiec 2025 roku) wszystkie opisane powyżej formy post-treningu obsługiwały wyłącznie modele o otwartych wagach. Jeżeli korzystasz z modelu hostowanego, sprawdź aktualną dokumentację dostawcy, aby się upewnić, czy dany wzorzec wymagający post-treningu jest dostępny — albo wybierz model, który zapewnia potrzebne możliwości.

## Dostrajanie modelu wiodącego

Firmy takie jak OpenAI i Anthropic, a także główni dostawcy chmury pokroju AWS i Google Cloud, uprościły proces post-treningu modeli wiodących z użyciem SFT. Można przesłać zbiór danych treningowych w postaci par wejście – wyjście i uruchomić proces dostrajania, którego efektem jest punkt końcowy modelu dostrojonego adapterami. Taki model można następnie wykorzystywać dokładnie tak samo jak model bazowy.

Zilustrujemy to na przykładzie modeli z rodziny GPT firmy OpenAI, choć dostrajanie modeli Anthropic Claude w Amazon Bedrock czy Gemini firmy Google w Vertex AI wygląda bardzo podobnie. Gdy przygotujesz pary treningowe, czyli zestawy (wejście – wyjście) — potrzebujesz co najmniej stu takich par, a najlepiej kilku tysięcy — zapisz je w pliku w formacie JSON. Następnie możesz wczytać zbiór treningowy w następujący sposób:

```
training_file = client.files.create(
    file=open("training_data.jsonl", "rb"),
    purpose="fine-tune"
)
```

Po załadowaniu danych możesz uruchomić zadanie dostrajania:

```
job = client.fine_tuning.jobs.create(
    training_file=training_file.id,
    model="gpt-3.5-turbo"  # Model bazowy do dostrojenia
)
```

Proces dostrajania potrwa od kilku minut do kilku godzin, w zależności od rozmiaru zbioru treningowego i wybranego modelu. Po zakończeniu treningu możesz odczytać identyfikator dostrojonego modelu:

```
job_status = client.fine_tuning.jobs.retrieve(job.id)
if job_status.status == 'succeeded':
    print(f"Identyfikator modelu: {job_status.fine_tuned_model}")
```

Wynikiem będzie identyfikator modelu w postaci:

```
ft:<MODEL BAZOWY>-0125:<NAZWA ORGANIZACJI>::<ID ZADANIA>
```

Użyj tego identyfikatora w API inferencyjnym, aby wywoływać model po dostrojeniu. Interfejs API pozostaje taki sam jak w przypadku modelu bazowego:

```
completion = client.chat.completions.create(
    model=job_status.fine_tuned_model,  # Użyj dostrojonego modelu
    messages=messages
)
print(completion.choices[0].message.content)
```

Dostrajanie modelu o otwartych wagach

Platforma Unsloth.ai umożliwia dostrajanie i trenowanie modeli LLM o otwartych wagach, takich jak Gemma czy Llama. Unsloth możesz uruchomić na własnym sprzęcie albo skorzystać z zarządzanych usług dostrajania oferowanych przez tę platformę.

Aby dostroić 4-bitowo skwantyzowaną wersję modelu Llama 3, zacznij od wczytania modelu oraz jego *tokenizera* — klasy, która dzieli sekwencję tekstu wejściowego na tokeny oczekiwane przez model:

```
from unsloth import FastLanguageModel
max_seq_length = 2048
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name="unsloth/Meta-Llama-3.1-8B-bnb-4bit",
    max_seq_length=max_seq_length,
    load_in_4bit=True,
    dtype=None,
)
```

Następnie dołącz do bazowego modelu Llama zestaw wag adaptera:

```
model = FastLanguageModel.get_peft_model(
    model,
    r=16,
    target_modules=["q_proj", "k_proj", "v_proj", "up_proj", "down_proj",
                    "o_proj", "gate_proj"],
    use_gradient_checkpointing="unsloth"
)
```

W poprzednim kodzie określono *rangę* (czyli *rozmiar macierzy*) warstwy LoRA oraz konkretne warstwy modelu, do których ma ona zostać zastosowana. Ten sam kod nakłada LoRA również na mechanizmy uwagi — macierze Q, K i V — a także na różne warstwy projekcyjne.

Zakładając, że pary wejście – wyjście są zapisane w jednym z formatów obsługiwanych przez Unsloth (*https://oreil.ly/bYfnX*), możesz wczytać zbiór danych za pomocą następującego kodu:

```
dataset = load_dataset("...", split="train")
dataset = dataset.map(apply_template, batched=True)
```

Następnie, na maszynie wyposażonej w wystarczająco wydajne procesory GPU, możesz uruchomić proces treningu w ten sposób:

```
trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=dataset,
    dataset_text_field="text",
)
trainer.train()
```

Po zakończeniu treningu możesz zapisać model i przesłać go do serwisu Hugging Face. Na tym etapie, przed opublikowaniem, możesz scalić model bazowy z warstwą adapterów w jeden model:

```
model.save_pretrained_merged("model", tokenizer, save_method="merged_16bit")
model.push_to_hub_merged("...", tokenizer, save_method="merged_16bit")
```

Tak przygotowanego modelu możesz następnie używać dokładnie tak samo jak modelu bazowego.

Kwestie praktyczne

Dostrajanie modeli daje wyjątkową możliwość dostosowania LLM do specyficznych potrzeb domenowych, ale zwiększa też złożoność. Zanim się na nie zdecydujesz, upewnij się, że korzyści równoważą dodatkowe koszty i zagrożenia.

Wymagania dotyczące danych

Zamiast kilku przykładów w kontekście potrzebujesz większej liczby próbek (ponad sto), aby dostosować zakres zadań, z którymi LLM ma sobie radzić. Wymaga to wcześniejszego zebrania danych. Jeśli na razie nie masz na to czasu lub zasobów, zacznij od uczenia w kontekście, zbieraj dane w trakcie, a dostrajanie wykonaj później. W ten sposób poprawisz jakość i uzyskasz bardziej stabilne, przewidywalne wyniki.

Katastrofalne zapominanie

Dostrajanie modeli LLM może prowadzić do *katastrofalnego zapominania*: model zaczyna nadmiernie trzymać się przykładów z dostrajania i traci część wiedzy wcześniej nabytej. To uderza w największą przewagę LLM — szeroką wiedzę. Ograniczysz ryzyko, jeśli użyjesz małego zbioru, wykonasz tylko kilka epok i dobierzesz właściwe tempo uczenia (parametr *learning rate*). Zwykle warto zacząć od wartości, na której skończył się wstępny trening (najczęściej około 1e-5).

Dodatkowa złożoność

Przed wdrożeniem modelu do produkcji musisz ocenić jakość jego działania i sprawdzić, czy model nie przejął niepożądanych wzorców, na przykład stronniczego języka. Gdy pojawia się nowa wersja modelu bazowego, musisz powtórzyć dostrajanie i ponownie przeprowadzić cały proces walidacji. Dodatkowo trzeba śledzić pochodzenie danych treningowych i walidacyjnych. To wszystko wymaga dyscypliny operacyjnej. Uczenie w kontekście jest pod tym względem znacznie prostsze, bo zwykle wystarczy kilka przykładów w prompcie.

Dodatkowe koszty

Tacy dostawcy jak OpenAI często naliczają wyższą cenę za tokeny podczas wnioskowania z użyciem modeli dostrojonych w porównaniu z modelami standardowymi. Wyższa stawka wynika z tego, że prompty do modeli dostrojonych mogą być znacznie krótsze, a mimo to dają porównywalną albo lepszą jakość. Ponieważ rozliczenia opierają się na tokenach wejściowych i wyjściowych, dostawcy podnoszą stawki, aby pokryć koszty hostowania modeli dostrojonych.

Z kolei dostrajanie modeli otwartych może obniżyć koszty wnioskowania, ale płacisz za GPU podczas treningu. W zależności od bazowego LLM może to kosztować tylko kilka dolarów, ale koszty potrafią szybko urosnąć do setek dolarów, zależnie od wersji modelu.

Jak zorganizowana jest pozostała część książki?

W dalszej części książki opisano 32 wzorce projektowe, ułożone w ośmiu rozdziałach. Poznasz metody sterowania wyjściami modelu, wzmacniania pozyskiwania wiedzy, poprawy rozumowania, podnoszenia niezawodności, uruchamiania działań, optymalizacji wydajności i wdrażania zabezpieczeń. Każdy podrozdział poświęcony określonemu wzorcowi zawiera czytelny opis problemu, podejście do rozwiązania, praktyczne scenariusze użycia oraz przykłady kodu. Mamy nadzieję, że ta książka pomoże Ci budować solidne i skuteczne aplikacje GenAI.

W rozdziale 2. pokazujemy, jak kontrolować styl i format treści generowanych przez AI — jest to kluczowa umiejętność z punktu widzenia spójności marki, poprawności oraz zgodności z wymaganiami. Przedstawiamy sposób wdrażania Logits Masking (wzorzec 1.), który pozwala wymuszać zgodność tekstu z określonymi regułami stylu przez przechwycenie procesu generowania na etapie próbkowania. Podrozdział dotyczący Grammar (wzorzec 2.) wyjaśnia, jak ograniczać wyniki do ściśle zdefiniowanych formatów lub schematów danych z użyciem formalnych specyfikacji gramatycznych. Style Transfer (wzorzec 3.) pokazuje, jak przekształcać treści tak, aby odpowiadały określonemu tonowi, z wykorzystaniem uczenia typu few-shot lub dostrajania. W części poświęconej Reverse Neutralization (wzorzec 4.) opisujemy podejście polegające na generowaniu treści neutralnej, a następnie jej transformacji do wyspecjalizowanego stylu. Na koniec opisujemy Content Optimization (wzorzec 5.), który przedstawia metody wyznaczania optymalnego stylu poprzez systematyczne porównania i strojenie preferencji, co ma szczególne znaczenie w marketingu, reklamie oraz materiałach edukacyjnych, gdzie skuteczne czynniki stylistyczne nie są od razu oczywiste.

W rozdziałach 3. i 4. poznasz wzorce, które pomogą Ci budować systemy AI korzystające z zewnętrznych źródeł wiedzy, aby obejść podstawowe ograniczenia, takie jak odcięcie od wiedzy (ang. *knowledge cutoff*), brak dostępu do danych poufnych i halucynacje. Zaczniemy od Basic RAG (wzorzec 6.), który uczy, jak uzyskiwać odpowiedzi AI oparte na informacjach pobranych z baz wiedzy. Semantic Indexing (wzorzec 7.) pokazuje, jak uchwycić znaczenie w różnych typach mediów dzięki osadzeniom (ang. *embeddings*), a więc jak wyjść poza proste dopasowanie słów kluczowych. Indexing at Scale (wzorzec 8.) przedstawia techniki zarządzania przestarzałymi lub sprzecznymi informacjami z wykorzystaniem metadanych, filtrowania i ponownego rangowania. Index-Aware Retrieval (wzorzec 9.) wprowadza bardziej zaawansowane metody, takie jak odpowiedzi hipotetyczne, rozszerzanie zapytań oraz GraphRAG, które poprawiają jakość wyszukiwania. Node Postprocessing (wzorzec 10.) pokazuje, jak radzić sobie z treściami nieistotnymi i niejednoznacznymi encjami dzięki ponownemu rangowaniu oraz kompresji kontekstowej. Następnie omawiamy Trustworthy Generation (wzorzec 11.), czyli podejścia pozwalające utrzymać zaufanie użytkowników mimo nieuniknionych błędów. Na końcu Deep Search (wzorzec 12.) przedstawia iteracyjne procesy złożonego pozyskiwania informacji, które omijają ograniczenia okna kontekstu i umożliwiają rozumowanie wieloetapowe (ang. *multihop reasoning*).

W rozdziale 5. omawiamy techniki, które wzmacniają rozumowanie i wyspecjalizowane kompetencje modeli językowych. Poznasz Chain of Thought (CoT) (wzorzec 13.), który pozwala modelom rozbijać złożone problemy na kroki pośrednie i wyraźnie poprawia wyniki w zadaniach matematycznych oraz wnioskowaniu logicznym. Tree of Thoughts (ToT), wzorzec 14., nauczy Cię wdrażać podejścia oparte na przeszukiwaniu drzewa w problemach, które wymagają eksploracji wielu ścieżek rozwiązań — to podejście świetnie sprawdza się w planowaniu i myśleniu strategicznym. Adapter Tuning (wzorzec 15.) pokazuje, jak skutecznie specjalizować modele LLM: trenujesz niewielkie, dołączane warstwy sieci neuronowej, a wagi modelu bazowego pozostają zamrożone. Dzięki temu specjalizacja jest możliwa nawet przy ograniczonych danych (od 100 do 10 000 przykładów). Evol-Instruct (wzorzec 16.) wyjaśnia, jak wydajnie generować wysokiej jakości zbiory do strojenia instrukcji przez wieloetapową ewolucję instrukcji. To podejście pozwala uczyć modele nowych zadań domenowych bez konieczności ręcznego wykonywania wielu operacji na danych.

W rozdziale 6. zapoznasz się ze wzorcami budowania bardziej niezawodnych systemów AI, którym można zaufać w produkcji. Poznasz LLM-as-Judge (wzorzec 17.) i nauczysz się oceniać możliwości GenAI przez szczegółową, wielowymiarową informację zwrotną. To podstawowa kompetencja przy porównywaniu modeli i śledzeniu postępów. Reflection (wzorzec 18.) pokazuje, jak umożliwić modelom korektę wcześniejszych odpowiedzi na podstawie opinii, co znacząco poprawia niezawodność w złożonych zadaniach. Dependency Injection (wzorzec 19.) prezentuje techniki niezależnego rozwijania i testowania komponentów łańcucha LLM, co ułatwia utrzymanie i niezawodność systemu. Prompt Optimization (wzorzec 20.) przedstawia sposoby systematycznego ustawiania i aktualizowania promptów przez optymalizację wykonywaną na zestawach przykładów. Takie podejście zmniejsza koszty utrzymania przy zmianach zależności i zapewnia stabilną jakość w czasie.

W rozdziale 7. omawiamy sposoby zmiany systemów AI z pasywnych dostawców informacji w aktywnych agentów zdolnych do działania. Opanujesz Tool Calling (wzorzec 21.) i nauczysz się łączyć LLM z API programów. Model może wtedy wywoływać funkcje z właściwymi parametrami i wykorzystywać wyniki w odpowiedziach. Zapewnia to dostęp do danych w czasie rzeczywistym, integracje z systemami firmowymi i pozwala wykonywać złożone obliczenia. Code Execution (wzorzec 22.) pokaże sposób wykorzystania LLM do generowania kodu, który wykona system zewnętrzny — to podejście sprawdza się przy tworzeniu wizualizacji, opisywaniu ilustracji oraz aktualizacji baz danych. Multiagent Collaboration (wzorzec 23.) nauczy Cię projektować systemy wyspecjalizowanych agentów, zorganizowanych na wzór struktur organizacyjnych. Takie systemy pozwalają realizować złożone rozumowanie, wieloetapowe rozwiązywanie problemów, współtworzenie treści i systemy samodoskonalące, które potrafią prowadzić dłuższe interakcje bez udziału człowieka.

W rozdziale 8. poznasz wzorce wdrażania GenAI z uwzględnieniem realnych ograniczeń związanych z kosztami, opóźnieniami i zasobami obliczeniowymi. Model SLM (ang. *Small Language Model*) — wzorzec 24. — pokazuje, jak wykorzystywać mniejsze, bardziej efektywne modele, które działają na urządzeniach krańcowych lub w środowiskach o ograniczonych zasobach, a mimo to zapewniają akceptowalną jakość w konkretnych zadaniach. Prompt Caching (wzorzec 25.) to prezentacja technik ograniczania powtarzalnych obliczeń i wywołań API, co istotnie obniża koszty w przypadku często żądanych treści. Inference Optimization (wzorzec 26.) wyposaży Cię w metody maksymalizacji przepustowości i minimalizacji opóźnień dzięki takim technikom, jak spekulatywne dekodowanie (ang. *speculative decoding*), ciągłe przetwarzanie wsadowe (ang. *continuous batching*) i kompresja promptów. Degradation Testing (wzorzec 27.) wyjaśnia, jak systematycznie oceniać jakość modelu w różnych scenariuszach wdrożeniowych i utrzymywać stabilny poziom w czasie. Long-Term Memory (wzorzec 28.) prezentuje, jak przechowywać historię użytkownika i dynamicznie stosować personalizację.

Rozdział 9. zawiera opis wzorców, które zapewniają bezpieczne i etyczne działanie aplikacji GenAI oraz utrzymanie ich w odpowiednich granicach. Template Generation (wzorzec 29.) pokazuje, jak przygotować i zweryfikować szablony, które w czasie wnioskowania wymagają wyłącznie deterministycznej podmiany tekstu. To podejście sprawdza się w komunikacji masowej, gdzie ręczna weryfikacja nie wchodzi w grę. Assembled Reformat (wzorzec 30.) nauczy Cię rozdzielać tworzenie treści na kroki o niskim ryzyku: najpierw bezpieczne zestawienie danych, potem atrakcyjne formatowanie. Dzięki temu spada ryzyko nieścisłości i halucynacji. Self-Check (wzorzec 31.) opisuje, jak wykorzystywać prawdopodobieństwa tokenów do opłacalnego wykrywania potencjalnych halucynacji w odpowiedziach faktograficznych. Guardrails (wzorzec 32.) to kompleksowe podejście do opakowywania wywołań LLM warstwami pre- i postprocessingu, które egzekwują ograniczenia bezpieczeństwa, prywatności, moderacji treści i zgodności z przepisami. Jest to niezbędne zawsze wtedy, gdy aplikacja może stać się celem nadużyć albo wrogich działań.

Na koniec, rozdział 10. omawia, jak łączyć wzorce z pierwszych dziewięciu rozdziałów w produkcyjną aplikację agentową.

Skorowidz

Od prototypu do produkcji — 32 sprawdzone wzorce dla aplikacji GenAI

Generatywna sztuczna inteligencja rewolucjonizuje pracę nad tworzeniem aplikacji. Modele bazowe takie jak GPT, Claude, Gemini czy Llama umożliwiają szybkie budowanie demonstracji, lecz rzeczywiste wdrożenia mierzą się z halucynacjami, niespójnymi odpowiedziami, lukami w wiedzy i problemami z niezawodnością. Książka przedstawia 32 wzorce projektowe wypracowane przez praktyków wdrażających systemy GenAI na dużą skalę. To nie teoria, lecz konkretne recepty na powtarzalne problemy, z którymi zmaga się każdy inżynier AI budujący aplikacje gotowe do produkcji.

Autorzy prowadzą czytelnika przez cały proces inżynierii AI — od kontrolowania stylu i formatu treści, przez integrację wiedzy zewnętrznej z zaawansowanym RAG, wzmacnianie zdolności wnioskowania, budowę niezawodnych systemów, tworzenie agentów zdolnych do działania, aż po optymalizację wdrożeń i implementację zabezpieczeń. Każdy wzorzec jest opisany według jednolitego schematu: problem, sprawdzone rozwiązanie, kompletny przykład kodu, omówienie alternatyw i kompromisów implementacyjnych. Książka łączy osiągnięcia najnowszych badań z praktycznym doświadczeniem ekspertów i dostarcza gotowe wskazówki dla osób tworzących aplikacje oparte na modelach bazowych.

W książce:

- RAG i zarządzanie wiedzą
- Wzorce agentowe
- Optymalizacja i skalowanie
- Zwiększanie niezawodności
- Zarządzanie stylem
- Zabezpieczenia
- Rozszerzanie możliwości

Valliappa (Lak) Lakshmanan jest współzałożycielem i CTO firmy Obin.ai. Wcześniej pełnił funkcję dyrektora do spraw rozwiązań AI w Google. Współautor kilku książek, w tym *Wzorce projektowe uczenia maszynowego*. Otrzymał tytuł Fellow American Meteorological Society za pionierskie zastosowanie ML w prognozowaniu pogody.

Hannes Hapke jest głównym inżynierem uczenia maszynowego w Digits. Otrzymał tytuł Google Developer Expert w dziedzinie ML, jest członkiem Developer Advisory Board Google. Współautor książek *Przetwarzanie języka naturalnego w akcji* i *Building Machine Learning Pipeline*.

KOD KORZYŚCI
Sięgnij po więcej!

helion.pl

HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-289-3820-5
9 788328 938205

Cena: 129,00 zł