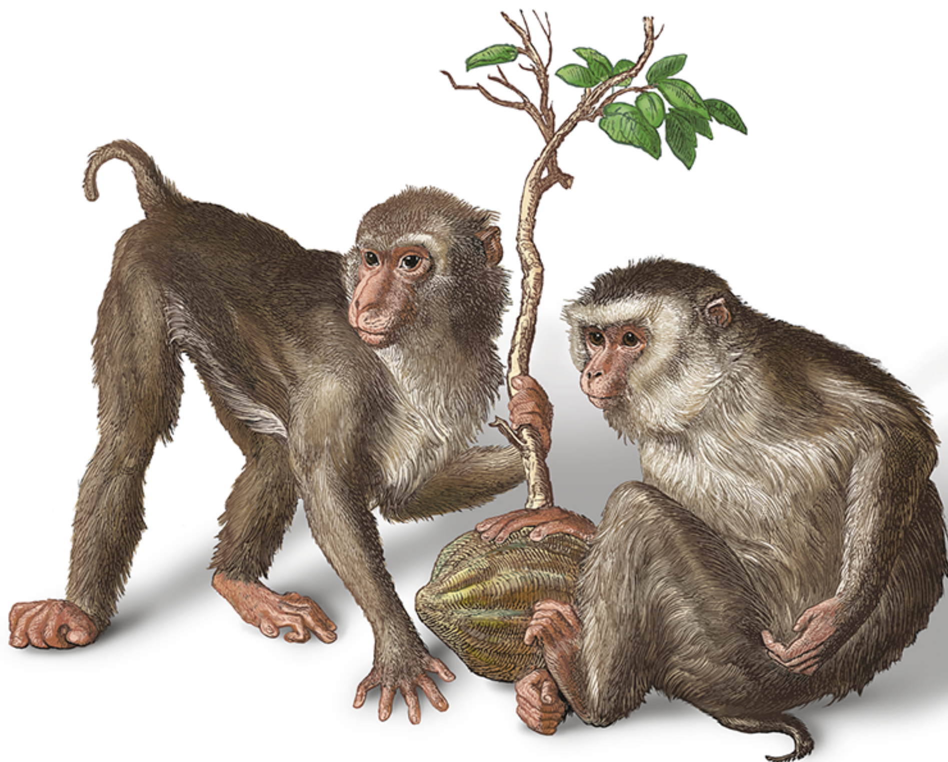
O'REILLY

Aplikacje oparte na agentach AI

Projektowanie i wdrażanie systemów wieloagentowych



Michael Albada

Tytuł oryginału: Building Applications with AI Agents: Designing and Implementing Multiagent Systems

Tłumaczenie: Grzegorz Werner

ISBN: 978-83-289-3682-9

© 2026 Helion S.A.

Authorized Polish translation of the English edition of *Building Applications with AI Agents*
ISBN 9781098176501 © 2025 Advance AI LLC.

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.

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)

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

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

Przedmowa 11

1. Wprowadzenie do agentów 18
Definicja agenta AI 18
Rewolucja treningu wstępnego 19
Rodzaje agentów 20
Wybór modelu 22
Od operacji synchronicznych do asynchronicznych 23
Zastosowania praktyczne i przypadki użycia 24
Przepływy pracy i agenty 25
Zasady tworzenia skutecznych systemów agentowych 28
Organizacja pracy pod kątem efektywnego budowania systemów agentowych 29
Platformy agentowe 30
LangGraph 30
AutoGen 31
CrewAI 31
OpenAI Agents Software Development Kit (SDK) 31
Podsumowanie 32

2. Projektowanie systemów agentowych 33
Nasz pierwszy system agentowy 33
Główne komponenty systemów agentowych 36
Wybór modelu 36
Narzędzia 40
Projektowanie funkcji pod kątem konkretnych zadań 40
Integracja narzędzi i modularność 41
Pamięć 41
Pamięć krótkotrwała 41
Pamięć długotrwała 42
Zarządzanie pamięcią i wyszukiwanie danych 42

Kup ksi k Pole ksi k

Orkiestracja 42
Kompromisy projektowe 43
Efektywność — kompromis między szybkością a dokładnością 43
Skalowalność — projektowanie skalowalnych systemów agentowych 44
Niezawodność — zapewnianie stabilnego i spójnego działania agentów 45
Koszty — równowaga między wydajnością a wydatkami 46
Architekturalne wzorce projektowe 48
Architektury jednoagentowe 48
Architektury wieloagentowe — współpraca, przetwarzanie równoległe i koordynacja 48
Najlepsze praktyki 50
Projektowanie iteracyjne 50
Strategia ewaluacji 51
Testowanie w warunkach rzeczywistych 53
Podsumowanie 55

3. Projektowanie wrażeń użytkowników systemów agentowych 56
Modalności interakcji 57
Interfejsy tekstowe 57
Interfejsy graficzne 60
Interfejsy głosowe 64
Interfejsy wideo 67
Łączenie modalności w celu zapewnienia spójnych wrażeń 68
Suwak autonomii 69
Synchroniczne i asynchroniczne rozwiązania agentowe 71
Zasady projektowania rozwiązań synchronicznych 72
Zasady projektowania rozwiązań asynchronicznych 72
Równowaga między aktywnym a natrętnym zachowaniem agenta 73
Podtrzymywanie kontekstu i ciągłość 74
Utrzymywanie stanu między interakcjami 75
Personalizacja i dostosowywanie 76
Komunikowanie możliwości agentów 76
Przekazywanie pewności i niepewności 78
Proszenie użytkowników o wskazówki i informacje 78
Eleganckie radzenie sobie z błędami 79
Zaufanie w projektowaniu interakcji 80
Podsumowanie 82

4. Używanie narzędzi 84
Podstawy LangChain 85
Narzędzia lokalne 86
Narzędzia oparte na API 88
Wtyczki 91

Kup ksi k Pole ksi k

Model Context Protocol 94
Narzędzia stanowe 97
Zautomatyzowane tworzenie narzędzi 98
Modele podstawowe jako twórcy narzędzi 98
Generowanie kodu w czasie rzeczywistym 99
Konfiguracja użycia narzędzi 100
Podsumowanie 101

5. Orkiestracja ... 102
Typy agentów 103
Agenty odruchowe 103
Agenty ReAct 103
Agenty planistyczno-wykonawcze 104
Agenty dekompozycyjne 104
Agenty refleksyjne 105
Agenty do badań pogłębionych 105
Wybór narzędzi 106
Standardowy wybór narzędzi 107
Semantyczny wybór narzędzi 110
Hierarchiczny wybór narzędzi 113
Wykonywanie narzędzi 117
Topologie narzędzi 118
Wykonywanie pojedynczych narzędzi 118
Równoległe wykonywanie narzędzi 119
Łańcuchy 120
Grafy 121
Inżynieria kontekstu 124
Podsumowanie 125

6. Wiedza i pamięć ... 127
Podstawowe podejścia do zarządzania pamięcią 128
Zarządzanie oknami kontekstowymi 128
Tradycyjne wyszukiwanie pełnotekstowe 129
Pamięć semantyczna i magazyny wektorowe 130
Wprowadzenie do wyszukiwania semantycznego 131
Budowanie pamięci semantycznej z wykorzystaniem magazynów wektorowych 131
Generowanie wspomagane wyszukiwaniem 133
Semantyczna pamięć doświadczeniowa 134
GraphRAG 135
Używanie grafów wiedzy 135
Tworzenie grafów wiedzy 136
Zalety i wady dynamicznych grafów wiedzy 142
Robienie notatek 144
Podsumowanie 145

Kup książkę Poleć książkę

7. Uczenie się w systemach agentowych 146
Uczenie nieparametryczne 146
Uczenie nieparametryczne na przykładach 146
Refleksja 148
Uczenie doświadczeniowe 152
Uczenie parametryczne — dostrajanie 156
Dostrajanie dużych modeli podstawowych 157
Potencjał małych modeli 161
Dostrajanie nadzorowane 163
Bezpośrednia optymalizacja preferencyjna 168
Uczenie ze wzmacnianiem z weryfikowalnymi nagrodami 171
Podsumowanie 172

8. Od jednego agenta do wielu 173
Ile agentów potrzebuję? 173
Scenariusze jednoagentowe 173
Scenariusze wieloagentowe 179
Roje 186
Zasady dodawania agentów 187
Koordynacja między agentami 189
Koordynacja demokratyczna 189
Koordynacja menedżerska 190
Koordynacja hierarchiczna 191
Metody aktor-krytyk 191
Automatyczne projektowanie systemów agentowych 193
Techniki komunikacyjne 197
Komunikacja lokalna a rozproszona 197
Protokół Agent-to-Agent 198
Brokery komunikatów i magistrale zdarzeń 201
Platformy aktorowe — Ray, Orleans i Akka 204
Mechanizmy orkiestracji i przepływu pracy 207
Zarządzanie stanem i trwałością danych 209
Podsumowanie 211

9. Walidacja i pomiary 213
Pomiary systemów agentowych 213
Pomiar to podstawa wszystkiego 214
Włączanie ewaluacji w cykl wytwarzania oprogramowania 214
Tworzenie i skalowanie zbiorów ewaluacyjnych 215
Ewaluacja komponentów 217
Ewaluacja narzędzi 217
Ewaluacja planowania 218

Kup książkę Poleć książkę

Ewaluacja pamięci 219
Ewaluacja uczenia 221
Ewaluacja holistyczna 221
Skuteczność w scenariuszach kompleksowych 222
Konsekwencja 224
Spójność 225
Halucynacje 226
Obsługa nieoczekiwanych danych wejściowych 227
Przygotowanie do wdrożenia 228
Podsumowanie 229

10. Monitorowanie w środowisku produkcyjnym 230
Monitorowanie jako źródło wiedzy 231
Stosy monitorowania 234
Grafana z OpenTelemetry, Loki i Tempo 234
Stos ELK (Elasticsearch, Logstash/Fluentd, Kibana) 235
Arize Phoenix 235
SigNoz 236
Langfuse 237
Wybór odpowiedniego stosu technologii 237
Instrumentacja OTel 238
Wizualizacja i alarmowanie 240
Wzorce monitorowania 242
Wdrożenie równoległe 242
Wdrożenia kanarkowe 243
Gromadzenie śladów regresji 243
Samonaprawiające się agenty 243
Opinie użytkowników jako sygnał obserwowalności 244
Przesunięcia rozkładów 244
Odpowiedzialność za miary i nadzór międzyfunkcyjny 247
Podsumowanie 249

11. Pętle doskonalenia 250
Potoki informacji zwrotnej 252
Automatyczne wykrywanie problemów i analiza przyczyn źródłowych 257
Przegląd z udziałem człowieka 258
Dopracowywanie promptów i narzędzi 261
Agregowanie ulepszeń i określanie ich priorytetów 266
Eksperymentowanie 267
Wdrożenia równoległe 268
Testy A/B 269
Bayesowski bandyta 271

Kup ksi k Pole ksi k

Ciągłe uczenie 272
Uczenie w kontekście 273
Ponowny trening w trybie offline 274
Podsumowanie 275

12. Zabezpieczanie systemów agentowych ... 277
Unikatowe zagrożenia związane z systemami agentowymi 278
Nowe wektory zagrożeń 280
Zabezpieczanie modeli podstawowych 281
Techniki defensywne 282
Red teaming 285
Modelowanie zagrożeń z użyciem MAESTRO 287
Ochrona danych w systemach agentowych 289
Prywatność danych i szyfrowanie 290
Pochodzenie i integralność danych 291
Zarządzanie danymi wrażliwymi 293
Zabezpieczanie agentów 295
Środki bezpieczeństwa 295
Ochrona przed zagrożeniami zewnętrznymi 296
Ochrona przed awariami wewnętrznymi 299
Podsumowanie 303

13. Współpraca ludzi i agentów ... 304
Role i autonomia 304
Zmieniająca się rola człowieka w systemach agentowych 304
Uzgadnianie oczekiwań interesariuszy i zachęcanie do korzystania z agentów 306
Skalowanie współpracy 307
Zakres działania agentów a role organizacyjne 309
Współdzielona pamięć i granice kontekstu 310
Zaufanie, nadzór i zgodność z przepisami 312
Cykl zaufania 312
Modele rozliczalności 313
Projektowanie i nadzorowanie eskalacji 316
Prywatność i zgodność z przepisami 317
Podsumowanie: przyszłość zespołów złożonych z ludzi i agentów 319

Słowniczek ... 321

Kup książkę Poleć książkę

ROZDZIAŁ 2.

Projektowanie systemów agentowych

Większość praktyków przystępujących do budowy systemów agentowych nie zaczyna od wielkiego dokumentu projektowego. Wychodzą od zagmatwanego problemu, klucza API do modelu podstawowego i mgliście zarysowanej koncepcji tego, co mogłoby pomóc. Ten rozdział to krótki przewodnik, który pomoże Ci szybko rozpocząć pracę. Każde z wymienionych zagadnień omówię dokładniej dalej w tej książce (wiele z nich doczeka się własnych rozdziałów), ale tutaj zyskasz ogólne pojęcie o tym, jak projektować systemy agentowe, wszystko na konkretnym przykładzie zarządzania obsługą klienta na platformie e-commerce.

Nasz pierwszy system agentowy

Zacznijmy od problemu, który mamy rozwiązać. Zespół obsługi klienta codziennie otrzymuje dziesiątki lub setki wiadomości e-mail z prośbami o zwrot pieniędzy za uszkodzony kubek, anulowanie niewysłanego zamówienia lub zmianę adresu dostawy. Pracownik musi przeczytać każdą wiadomość, wyszukać zamówienie w systemie, wywołać odpowiednie API, a następnie napisać wiadomość potwierdzającą. Ten powtarzalny dwuminutowy proces nadaje się idealnie do automatyzacji — ale tylko wtedy, gdy wydzielimy odpowiedni fragment. Gdy zdamy sobie sprawę, że ludzie naciskają klawisze i klikają przyciski, często postępując zgodnie z regułami i wytycznymi, widzimy, że wiele z tych samych wzorców może być wykonywanych przez dobrze zaprojektowane systemy oparte na modelach podstawowych. Chcemy, aby nasz agent pobrał nieprzetworzoną wiadomość klienta wraz z rekordem zamówienia, zdecydował, które narzędzie wywołać (`issue_refund`, `cancel_order` lub `update_address_for_order`), uruchomił to narzędzie z prawidłowymi parametrami, a następnie wysłał krótką wiadomość potwierdzającą. Ten dwuetapowy przepływ pracy jest na tyle ograniczony, że można go szybko zbudować, na tyle wartościowy, że pozwoli użytkownikom zyskać więcej czasu, i na tyle bogaty, że pokaże inteligentne zachowanie. Do zbudowania agenta obsługującego ten scenariusz wystarczy kilka wierszy kodu:

```
from langchain.tools import tool
from langchain_openai.chat_models import ChatOpenAI
from langchain.schema import SystemMessage, HumanMessage, AIMessage
from langchain_core.messages.tool import ToolMessage
from langgraph.graph import StateGraph
```


Kup ksi k Pole ksi k

```python
# --1) Definiujemy nasze pojedyncze narzędzie biznesowe
@tool
def cancel_order(order_id: str) -> str:
    """Anuluj zamówienie, które nie zostało wysłane."""
    # (Tutaj wywołałbyś rzeczywiste API)
    return f"Zamówienie {order_id} zostało anulowane."

# --2) „Mózg" agenta: wywołać LLM, uruchomić narzędzie, ponownie wywołać LLM
def call_model(state):
    msgs = state["messages"]
    order = state.get("order", {"order_id": "UNKNOWN"})

    # Prompt systemowy mówi modelowi, co trzeba zrobić
    prompt = (
        f'''Jesteś agentem wsparcia w systemie ecommerce.
        ORDER ID: {order['order_id']}
        Jeśli klient prosi o anulowanie, wywołaj cancel_order(order_id)
        a następnie wyślij proste potwierdzenie.
        W przeciwnym razie po prostu odpowiedz normalnie.'''
    )
    full = [SystemMessage(prompt)] + msgs

    # Pierwszy przebieg LLM-a: decyduje, czy wywołać narzędzie
    AIMessage = ChatOpenAI(model="gpt-5", temperature=0)(full)
    out = [first]

    if getattr(first, "tool_calls", None):
        # Uruchamiamy narzędzie cancel_order
        tc = first.tool_calls[0]
        result = cancel_order(**tc["args"])
        out.append(ToolMessage(content=result, tool_call_id=tc["id"]))

        # Drugi przebieg LLM-a: generujemy tekst potwierdzenia
        AIMessage = ChatOpenAI(model="gpt-5", temperature=0)(full + out)
        out.append(second)

    return {"messages": out}

# -- 3) Łączymy wszystkie elementy w obiekcie StateGraph
def construct_graph():
    g = StateGraph({"order": None, "messages": []})
    g.add_node("assistant", call_model)
    g.set_entry_point("assistant")
    return g.compile()

graph = construct_graph()

if __name__ == "__main__":

    example_order = {"order_id": "A12345"}
    convo = [HumanMessage(content="Proszę anulować moje zamówienie A12345.")]
    result = graph.invoke({"order": example_order, "messages": convo})
    for msg in result["messages"]:
        print(f"{msg.type}: {msg.content}")
```

Świetnie — masz teraz działającego agenta do „anulowania zamówień". Zanim go rozbudujemy, zastanówmy się, *dlaczego* zaczęliśmy od tak prostego wycinka funkcjonalności. Wyznaczanie zakresu zawsze polega na szukaniu równowagi. Jeśli zawęzisz zadanie zbyt mocno — na przykład

Kup książkę Poleć książkę

tylko do anulowania zamówienia — pominiesz inne częste zapytania o zwrot pieniędzy czy zmianę adresu, a przez to rzeczywisty wpływ Twojego rozwiązania będzie ograniczony. Ale jeśli rozszerzysz zakres zbyt szeroko — „zautomatyzuj obsługę każdego zapytania" — utoniesz w przypadkach brzegowych takich jak spory rozliczeniowe, rekomendacje produktów czy rozwiązywanie problemów technicznych. Jeżeli natomiast sformułujesz zadanie zbyt ogólnie — „popraw satysfakcję klientów" — nigdy nie będziesz wiedział, czy odniosłeś sukces.

Zamiast tego skupmy się na jasnym, ograniczonym przepływie pracy — anulowaniu zamówień. W ten sposób zapewniamy konkretne dane wejściowe (wiadomość od klienta + rekord zamówienia), uporządkowane wyniki (wywołania narzędzi + potwierdzenia) oraz szybką pętlę informacji zwrotnych. Wyobraź sobie na przykład e-mail mówiący: „Proszę anulować moje zamówienie #B73973, znalazłem tańszą opcję gdzie indziej". Konsultant wyszukałby zamówienie, sprawdził, czy nie zostało wysłane, kliknął „Anuluj" i odesłał potwierdzenie. Przetłumaczenie tego na kod oznacza wywołanie funkcji `cancel_order(order_id="B73973")` i odesłanie prostej wiadomości potwierdzającej do klienta.

Teraz, gdy mamy agenta do anulowania zamówień, następne pytanie brzmi: czy rzeczywiście działa? W środowisku produkcyjnym nie wystarczy, żeby nasz agent się uruchamiał — chcemy wiedzieć, jak dobrze funkcjonuje, co robi poprawnie i w jakich sytuacjach zawodzi. W przypadku naszego agenta do anulowania zamówień interesują nas takie pytania jak:

- Czy agent wywołał właściwe narzędzie (`cancel_order`)?
- Czy przekazał odpowiednie parametry (poprawny identyfikator zamówienia)?
- Czy wysłał do klienta jasną, poprawną wiadomość potwierdzającą?

W naszym repozytorium open source znajdziesz pełny skrypt ewaluacyjny do automatyzacji tego procesu:

- Zbiór danych do oceny (*https://oreil.ly/GitHub-eval-set*)
- Skrypt do oceny wsadowej (*https://oreil.ly/GitHub-batch-eval*)

Oto minimalna, uproszczona wersja logiki do bezpośredniego testowania agenta:

```
# Minimalna ewaluacja
example_order = {"order_id": "B73973"}
convo = [HumanMessage(content='''Proszę anulować zamówienie #B73973.
    Znalazłem tańszą opcję gdzie indziej.''')]
result = graph.invoke({"order": example_order, "messages": convo})

assert any("cancel_order" in str(m.content) for m in result["messages"],
    "Narzędzie do anulowania zamówienia nie zostało wywołane")
assert any("cancelled" in m.content.lower() for m in result["messages"],
    "Brak komunikatu potwierdzającego")

print("✅ Agent przeszedł minimalną ewaluację.")
```

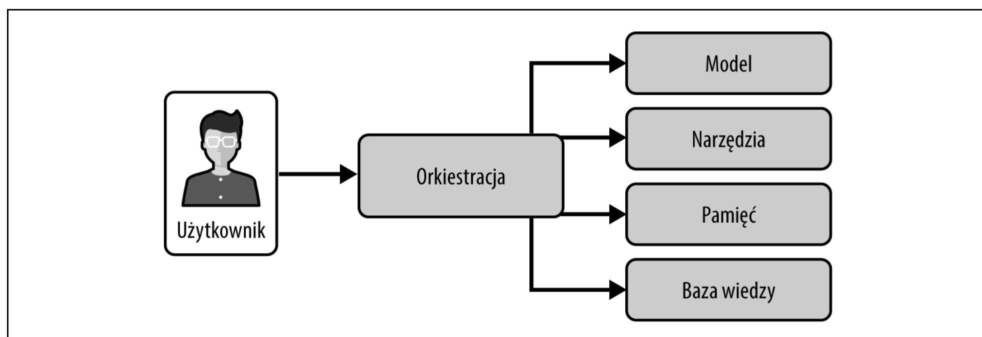
Ten fragment kodu sprawdza, czy narzędzie zostało wywołane, a potwierdzenie wysłane. Oczywiście rzeczywista ewaluacja sięgałaby znacznie głębiej: można mierzyć precyzję narzędzi, dokładność parametrów oraz ogólne wskaźniki powodzenia zadań na setkach przykładów, aby wychwycić przypadki brzegowe przed wdrożeniem agenta. Strategie i platformy ewaluacji omówimy w rozdziale 9, ale na razie zapamiętaj, że nieprzetestowanemu agentowi nie można ufać.

Kup ksi k Pole ksi k

Ponieważ oba kroki są zautomatyzowane przy użyciu dekoratorów `@tool`, pisanie testów dla rzeczywistych zgłoszeń staje się banalnie proste, a Ty natychmiast zyskujesz mierzalne wskaźniki, takie jak czułość narzędzia, dokładność parametrów oraz jakość potwierdzeń. Teraz, gdy zbudowaliśmy i oceniliśmy minimalnego agenta, przyjrzyjmy się kluczowym decyzjom projektowym, które będą kształtować jego możliwości i wpływ.

Główne komponenty systemów agentowych

Projektowanie skutecznego systemu agentowego wymaga głębokiego zrozumienia kluczowych komponentów, które umożliwiają agentom pomyślne wykonywanie powierzonych im zadań. Każdy z tych komponentów odgrywa istotną rolę w kształtowaniu możliwości agenta, jego efektywności oraz zdolności adaptacyjnych, począwszy od wyboru odpowiednich modeli, a skończywszy na wyposażeniu agenta w narzędzia, pamięć oraz możliwości planowania. Wszystkie te elementy muszą współgrać ze sobą, aby umożliwić agentowi funkcjonowanie w dynamicznych i złożonych środowiskach. W niniejszym podrozdziale szczegółowo omówię kluczowe komponenty — model podstawowy, narzędzia oraz pamięć — i wyjaśnię, jak ich wzajemne interakcje tworzą spójny system agentowy. Główne komponenty systemu agentowego pokazano na rysunku 2.1.



Rysunek 2.1. Główne komponenty systemu agentowego

Wybór modelu

Podstawą każdego systemu agentowego jest model, który pozwala agentowi na podejmowanie decyzji, interakcje i uczenie się. Wybór odpowiedniego modelu ma fundamentalne znaczenie — to on określa, jak agent interpretuje dane wejściowe, generuje wyniki i dostosowuje się do swojego środowiska. Decyzja ta wpływa na wydajność systemu, skalowalność, opóźnienia i koszty. Wybór właściwego modelu zależy od złożoności zadań agenta, charakteru danych wejściowych, ograniczeń infrastrukturalnych oraz kompromisów między uniwersalnością, szybkością i precyzją.

Ogólnie rzecz biorąc, wybór modelu rozpoczyna się od oceny złożoności zadań. Duże modele podstawowe — takie jak GPT-5 czy Claude Opus 4.1 — doskonale sprawdzają się w przypadku

Kup ksi k Pole ksi k

agentów działających w otwartych środowiskach, w których kluczowe są subtelne rozumienie, elastyczne rozumowanie i kreatywne generowanie treści. Modele te oferują imponującą zdolność do generalizacji i sprawdzają się doskonale w zadaniach związanych z niejednoznacznością, niuansami kontekstowymi czy wieloetapowymi procesami. Wiąże się to jednak z pewnymi kosztami — modele te wymagają znacznych zasobów obliczeniowych, często potrzebują infrastruktury chmurowej i wprowadzają większe opóźnienia. Najlepiej sprawdzają się w zastosowaniach takich jak asystenty osobiste, agenty badawcze czy systemy korporacyjne, które muszą radzić sobie z szerokim spektrum nieprzewidywalnych zapytań.

Natomiast mniejsze modele — takie jak destylowane warianty ModernBERT czy Phi-4 — często lepiej nadają się dla agentów wykonujących dobrze zdefiniowane, powtarzalne zadania. Modele te działają wydajnie na lokalnym sprzęcie, reagują szybko i są tańsze we wdrażaniu i utrzymaniu. Sprawdzają się dobrze w ustrukturyzowanych środowiskach, takich jak obsługa klienta, wyszukiwanie informacji czy etykietowanie danych, gdzie potrzebna jest precyzja, a kreatywność i elastyczność mają mniejsze znaczenie. Gdy kluczowe jest reagowanie w czasie rzeczywistym lub występują ograniczenia zasobów, mniejsze modele mogą przewyższać swoje większe odpowiedniki po prostu dlatego, że są bardziej praktyczne.

Coraz ważniejszym kryterium wyboru modelu staje się modalność. Współczesne agenty często muszą przetwarzać nie tylko tekst, lecz także obrazy, dźwięk czy dane strukturalne. Modele multimodalne, takie jak GPT-5 i Claude 4.1, umożliwiają agentom interpretowanie i łączenie różnorodnych typów danych — tekstu, obrazów, mowy i innych. Zwiększa to użyteczność agenta w dziedzinach takich jak opieka zdrowotna, robotyka i obsługa klienta, w których decyzje opierają się na integracji danych wejściowych w wielu postaciach. Z kolei modele obsługujące wyłącznie tekst pozostają idealne dla zastosowań czysto językowych, ponieważ oferują mniejszą złożoność i szybsze wnioskowanie w scenariuszach, w których dodatkowe modalności wnoszą niewielką wartość dodaną.

Kolejną kluczową kwestią jest otwartość i możliwość dostosowania. Modele open source, takie jak Llama i DeepSeek, zapewniają programistom pełną przejrzystość oraz możliwość dostrajania lub modyfikowania modelu według potrzeb. Ta elastyczność jest szczególnie ważna w zastosowaniach kładących nacisk na prywatność, regulowanych prawnie lub specjalistycznych. Modele open source można hostować w prywatnej infrastrukturze, dostosowywać do unikatowych przypadków użycia i wdrażać bez kosztów licencyjnych — choć wymagają większego nakładu pracy inżynieryjnej. Z kolei modele zastrzeżone, takie jak GPT-5, Claude i Cohere, oferują zaawansowane możliwości interakcji poprzez API i są dostarczane wraz z zarządzaną infrastrukturą, monitoringiem i optymalizacjami wydajności. Modele te są idealne dla zespołów dążących do szybkiego rozwoju i wdrożenia, choć możliwości ich dostosowywania są często ograniczone, a koszty mogą szybko rosnąć wraz z użytkowaniem.

Wybór między wstępnie wytrenowanym modelem ogólnego przeznaczenia a samodzielnie trenowanym modelem zależy od specyfiki dziedziny agenta. Modele wstępnie wytrenowane — szkolone na szerokich korpusach internetowych — sprawdzają się dobrze w ogólnych zadaniach językowych, szybkim prototypowaniu i scenariuszach, w których precyzja w danej dziedzinie nie jest kluczowa. Modele te można często lekko dostroić lub zaadaptować poprzez techniki formułowania promptów, aby osiągnąć dużą trafność minimalnym nakładem pracy.

Kup ksi k Pole ksi k

Jednak w wyspecjalizowanych dziedzinach — takich jak medycyna, prawo czy pomoc techniczna — samodzielnie trenowane modele mogą zapewnić znaczącą przewagę. Poprzez trening na wyselekcjonowanych, specyficznych dla konkretnej dziedziny zbiorach danych programiści mogą wyposażyć agentów w głębszą wiedzę dziedzinową i rozumienie kontekstu, co prowadzi do dokładniejszych i bardziej wiarygodnych wyników.

W rzeczywistych wdrożeniach szalę często przechylają kwestie kosztów i opóźnień. Duże modele zapewniają wysoką efektywność, ale są drogie w eksploatacji i mogą wprowadzać opóźnienia w odpowiadaniu. Kiedy jest to nie do przyjęcia, mniejsze modele lub skompresowane wersje większych modeli oferują lepszą równowagę. Wielu programistów przyjmuje strategie hybrydowe — zaawansowany model obsługuje najbardziej złożone zapytania, a uproszczony zajmuje się rutynowymi zadaniami. W niektórych systemach używa się dynamicznego routingu modeli, dzięki czemu każde żądanie jest oceniane i kierowane do najbardziej odpowiedniego modelu na podstawie złożoności lub pilności, co umożliwia systemom optymalizowanie zarówno kosztów, jak i jakości.

Centrum Badań nad Modelami Podstawowymi na Uniwersytecie Stanforda opublikowało holistyczną ewaluację modeli językowych — niezależne, rygorystyczne pomiary efektywności szerokiej gamy modeli. W tabeli 2.1 przedstawiono wybór wyników modeli językowych w teście porównawczym *Massive Multitask Language Understanding* (MMLU), powszechnie używanym narzędziu do oceny ogólnych zdolności tych modeli. Pomiary te nie są doskonałe, ale zapewniają wspólne ramy dla porównywania modeli. Ogólnie widać, że większe modele działają lepiej, ale zależność nie jest oczywista (niektóre modele działają lepiej, niż sugerowałby ich rozmiar). Do uzyskania wysokiej efektywności wymaganych jest znacznie więcej zasobów obliczeniowych.

Tabela 2.1. Wybrane modele otwarte według wydajności i rozmiaru

Model	Twórca	MMLU	Parametry (miliardy)	VRAM (model o pełnej precyzji, GB)	Przykładowy wymagany sprzęt
Llama 3.1 Instruct Turbo	Meta	56,1	8	20	RTX 3090
Gemma 2	Google	72,1	9	22,5	RTX 3090
NeMo	Mistral	65,3	12	24	RTX 3090
Phi-3	Microsoft	77,5	14,7	29,4	A100
Qwen1.5	Alibaba	74,4	32	60,11	A100
Llama 3	Meta	79,3	70	160	4xA100

Z drugiej strony oznacza to, że umiarkowanie dobre wyniki wydajności można uzyskać za ułamek kosztów. Jak widać w tabeli 2.1, modele liczące do około 14 miliardów parametrów można uruchamiać na pojedynczej karcie graficznej klasy konsumenckiej, takiej jak NVIDIA RTX 3090 z 24 GB pamięci wideo RAM. Powyżej tego progu prawdopodobnie będziesz potrzebować karty graficznej klasy serwerowej, takiej jak NVIDIA A100, dostępnej w wersjach 40 GB i 80 GB. Modele nazywane są „otwartymi” (ang. *open weight*), gdy ich architektura i wagi (parametry) zostały udostępnione publicznie za darmo, dzięki czemu każda osoba posiadająca odpowiedni

Kup ksi k Pole ksi k

sprzęt może wczytać model i używać go do wnioskowania bez płacenia za dostęp. Nie będziemy zagłębiać się w szczegóły doboru sprzętu, ale te wybrane otwarte modele pokazują różne poziomy wydajności przy różnych rozmiarach. Małe, otwarte modele poprawiają się w szybkim tempie i dostarczają coraz więcej inteligencji w mniejszych formatach. Choć nie sprawdzą się przy najtrudniejszych problemach, potrafią obsłużyć łatwiejsze, bardziej rutynowe zadania za ułamek ceny. W naszym przykładzie agenta wsparcia e-commerce wystarczy mały, szybki model — ale gdybyśmy rozszerzyli funkcjonalność o rekomendacje produktów czy eskalację opartą na analizie odczuć, większy model mógłby odblokować nowe możliwości.

Przyjrzyjmy się teraz kilku dużym, sztandarowym modelom. Warto zauważyć, że dwa z tych modeli — DeepSeek-v3 i Llama 3.1 Instruct Turbo 405B — zostały udostępnione jako modele otwarte, ale pozostałe nie. Te duże modele zazwyczaj wymagają co najmniej 12 kart graficznych w celu uzyskania rozsądnej wydajności, ale mogą potrzebować znacznie więcej. Duże modele są niemal zawsze używane na serwerach w dużych centrach danych. Twórcy modeli zwykle naliczają opłaty za dostęp na podstawie liczby tokenów wejściowych i wyjściowych. Zaletą tego rozwiązania jest to, że programista nie musi martwić się serwerami i wykorzystaniem kart graficznych, lecz może od razu rozpocząć tworzenie aplikacji. Tabela 2.2 pokazuje koszty modeli i ich wyniki w tym samym teście MMLU.

Tabela 2.2. Wybrane duże modele według wydajności i kosztów

Model	Twórca	MMLU	Względna cena za milion tokenów wejściowych	Względna cena za milion tokenów wyjściowych
DeepSeek-v3	DeepSeek	87,2	2,75	3,65
Claude 4 Opus Extended Thinking	Anthropic	86,5	75	125
Gemini 2.5 Pro	Google	86,2	12,5	25
Llama 3.1 Instruct Turbo 405B	Meta	84,5	1	1
o4-mini	OpenAI	83,2	5,5	7,33
Grok 3	xAI	79,9	15	25
Nova Pro	Amazon	82,0	4	5,33
Mistral Large 2	Mistral	80,0	10	10

W tabeli 2.2 ceny przedstawiono jako wielokrotność ceny za milion tokenów modelu Llama 3.1, który był najtańszy w momencie publikacji tej książki; Meta pobierała wówczas 0,20 dolara za milion tokenów wejściowych i 0,60 dolara za milion tokenów wyjściowych. Można zauważyć, że wydajność nie koreluje bezpośrednio z ceną. Warto również wiedzieć, że wyniki testów porównawczych stanowią przydatną wskazówkę, ale rzeczywiste rezultaty mogą się różnić w zależności od tego, jak dobrze te testy odpowiadają konkretnym zadaniom. Jeśli to możliwe, porównaj modele pod kątem własnych zadań i znajdź taki, który zapewnia najlepszy stosunek ceny do efektywności.

Kup ksi k Pole ksi k

Ostatecznie wybór modelu to nie jednorazowa decyzja, lecz strategiczny wybór projektowy, do którego należy wracać w miarę rozwoju możliwości agentów, potrzeb użytkowników i infrastruktury. Programiści muszą rozważać kompromisy między uniwersalnością a specjalizacją, wydajnością a kosztami, prostotą a rozszerzalnością. Poprzez staranne rozważenie złożoności zadań, modalności danych wejściowych, ograniczeń operacyjnych i potrzeb w zakresie dostosowywania zespoły mogą wybrać modele, które umożliwią agentom efektywne działanie, niezawodne skalowanie i precyzyjne wykonywanie zadań w rzeczywistym świecie.

Narzędzia

W systemach agentowych *narzędzia* to podstawowe komponenty umożliwiające agentom wykonywanie określonych działań i rozwiązywanie problemów. Narzędzia to funkcjonalne elementy składowe agenta, zapewniające mu zdolność do realizacji zadań oraz interakcji zarówno z użytkownikami, jak i innymi systemami. Skuteczność agenta zależy od zakresu i poziomu zaawansowania dostępnych narzędzi.

Projektowanie funkcji pod kątem konkretnych zadań

Narzędzia są zazwyczaj dostosowywane do zadań, które ma wykonywać agent. Podczas projektowania narzędzi programiści muszą się zastanowić, jak agent będzie działał w różnych warunkach i kontekstach. Dobrze zaprojektowany zestaw narzędzi pozwala agentowi precyzyjnie i wydajnie radzić sobie z różnorodnymi zadaniami. Narzędzia można podzielić na trzy główne kategorie:

Narzędzia lokalne

Działania wykonywane przez agenta w oparciu o wewnętrzną logikę i obliczenia, bez zewnętrznych zależności. Narzędzia lokalne często bazują na regułach lub obejmują wykonywanie predefiniowanych funkcji. Przykłady to obliczenia matematyczne, pobieranie danych z lokalnych baz danych czy proste podejmowanie decyzji na podstawie ustalonych reguł (np. decydowanie o zatwierdzeniu lub odrzuceniu wniosku według określonych kryteriów).

Narzędzia oparte na API

Narzędzia oparte na API umożliwiają agentom interakcję z zewnętrznymi usługami lub źródłami danych. Pozwalają rozszerzyć możliwości agenta poza lokalne środowisko poprzez pobieranie danych w czasie rzeczywistym lub wykorzystywanie systemów zewnętrznych. Na przykład asystent wirtualny może używać API do pobierania danych pogodowych, cen akcji czy aktualizacji z mediów społecznościowych, co pozwala mu udzielać bardziej kontekstowych, trafnych odpowiedzi na pytania użytkowników.

Model Context Protocol (MCP)

Narzędzia oparte na MCP umożliwiają agentom dostarczanie ustrukturyzowanego kontekstu w czasie rzeczywistym do modeli językowych przy użyciu protokołu Model Context Protocol (*https://oreil.ly/tSd_a*) — standardowego formatu przekazywania zewnętrznej

Kup ksi k Pole ksi k

wiedzy, pamięci i stanu do promptu modelu. W przeciwieństwie do tradycyjnych wywołań API wymagających pełnego cyklu wykonania, MCP pozwala agentom wstrzykiwać bogaty, dynamiczny kontekst — taki jak profile użytkowników, historia rozmów, stan świata czy metadane specyficzne dla zadania — bezpośrednio do procesu rozumowania modelu bez wywoływania oddzielnych narzędzi. Protokół ten jest szczególnie skuteczny w ograniczaniu nadmiarowego używania narzędzi, podtrzymywaniu stanu konwersacji i przekazywaniu świadomości sytuacyjnej do zachowań modelu.

Podczas gdy narzędzia lokalne umożliwiają agentom samodzielne wykonywanie zadań z użyciem wewnętrznej logiki i funkcji opartych na regułach, takich jak obliczenia czy pobieranie danych z lokalnych baz, narzędzia API pozwalają agentom łączyć się z zewnętrznymi usługami. Umożliwia to dostęp w czasie rzeczywistym do danych lub systemów zewnętrznych w celu zapewnienia kontekstowo trafnych odpowiedzi i rozszerzonej funkcjonalności.

Integracja narzędzi i modularność

Projektowanie modularne ma kluczowe znaczenie podczas tworzenia narzędzi. Każde narzędzie powinno być zaprojektowane jako samodzielny moduł, który można łatwo zintegrować lub wymienić w razie potrzeby. Takie podejście pozwala programistom aktualizować lub rozszerzać funkcje agenta bez konieczności przebudowywania całego systemu. Czatbot do obsługi klienta może na początku dysponować podstawowym zestawem narzędzi do obsługi prostych zapytań, a później można do niego dodać bardziej zaawansowane narzędzia (np. do rozstrzygania sporów czy zaawansowanego rozwiązywania problemów) bez zakłócania podstawowego działania agenta.

Pamięć

Pamięć stanowi kluczowy komponent umożliwiający agentom przechowywanie i odzyskiwanie informacji, dzięki czemu mogą podtrzymywać kontekst, uczyć się z poprzednich interakcji oraz stopniowo poprawiać proces podejmowania decyzji. Skuteczne zarządzanie pamięcią sprawia, że agenty mogą działać wydajnie w dynamicznych środowiskach i dostosowywać się do nowych sytuacji w oparciu o dane historyczne. Pamięć omówimy znacznie bardziej szczegółowo w rozdziale 6.

Pamięć krótkotrwała

Pamięć krótkotrwała odnosi się do zdolności agenta do przechowywania informacji istotnych dla bieżącego zadania lub rozmowy i zarządzania nimi. Ten rodzaj pamięci jest zwykle używany do podtrzymywania kontekstu podczas interakcji, co umożliwia agentowi podejmowanie spójnych decyzji w czasie rzeczywistym. Agent do obsługi klienta, który pamięta wcześniejsze zapytania użytkownika w ramach sesji, może udzielać bardziej trafnych i kontekstowych odpowiedzi, poprawiając tym samym wrażenia użytkownika.

Kup ksi k

Pole ksi k

Pamięć krótkotrwałą często implementuje się z użyciem *ruchomych okien kontekstowych*, które umożliwiają agentowi dostęp do najnowszych informacji przy jednoczesnym odrzucaniu przestarzałych danych. Jest to szczególnie przydatne w aplikacjach takich jak czatboty czy asystenty wirtualne, w których agent musi pamiętać niedawne interakcje, ale może zapomnieć starsze, nieistotne szczegóły.

Pamięć długotrwała

Pamięć długotrwała umożliwia agentom przechowywanie wiedzy i doświadczeń przez dłuższe okresy, co pozwala im podejmować przyszłe działania na podstawie wcześniejszych informacji. Ma to szczególne znaczenie dla agentów, które muszą się doskonalić z czasem lub zapewniać spersonalizowane wrażenia oparte na preferencjach użytkowników.

Pamięć długotrwałą często implementuje się z użyciem baz danych, grafów wiedzy lub dostrojonych modeli. Te struktury umożliwiają agentom przechowywanie danych ustrukturyzowanych (np. preferencje użytkowników, historyczne miary efektywności) i pobieranie ich w razie potrzeby. Agent monitorujący opiekę zdrowotną może utrzymywać długoterminowe informacje dotyczące parametrów życiowych pacjenta, co pozwala mu na wykrywanie trendów lub dostarczanie spostrzeżeń na podstawie historycznych danych.

Zarządzanie pamięcią i wyszukiwanie danych

Skuteczne zarządzanie pamięcią polega na organizowaniu i indeksowaniu przechowywanych danych w taki sposób, aby można je było łatwo wyszukać w razie potrzeby. Agenty korzystające z pamięci muszą umieć rozróżniać dane istotne od nieistotnych oraz szybko pobierać informacje. W niektórych przypadkach agenty potrzebują również sposobu na zapominanie pewnych informacji, aby uniknąć zaśmiecania pamięci przestarzałymi lub niepotrzebnymi szczegółami.

Agent rekomendacji e-commerce musi przechowywać preferencje użytkowników oraz historię wcześniejszych zakupów, aby dostarczać spersonalizowane rekomendacje. Musi jednak również nadawać pierwszeństwo najnowszym danym, żeby rekomendacje pozostawały trafne i dokładne pomimo zmieniających się preferencji użytkowników.

Orkiestracja

Orkiestracja to proces, który przekształca izolowane możliwości w kompleksowe rozwiązania — jest to logika łącząca, planująca i nadzorująca szereg umiejętności tak, aby każde działanie płynnie przechodziło w następne i zmierzało ku jasnemu celowi. Orkiestracja zasadniczo ocenia możliwe sekwencje wywołań narzędzi lub umiejętności, przewiduje ich prawdopodobne rezultaty i wybiera ścieżkę, która ma największe szanse powodzenia w zadaniach wieloetapowych — niezależnie od tego, czy chodzi o wyznaczenie optymalnej trasy dostawy uwzględniającej ruch, okna czasowe i dostępność pojazdów, czy też o złożenie skomplikowanego potoku przetwarzania danych.

Kup ksi k   Pole ksi k

Ponieważ warunki w rzeczywistym świecie mogą się zmienić w mgnieniu oka — nadchodzą nowe informacje, zmieniają się priorytety lub zasoby stają się niedostępne — orkiestrator musi stale monitorować zarówno postępy zadania, jak i środowisko, i w zależności od tego odpowiednio wstrzymywać lub przekierowywać przepływy pracy, aby utrzymać właściwy kurs. W wielu scenariuszach agenty budują plany stopniowo: wykonują kilka kroków, następnie ponownie oceniają sytuację i aktualizują pozostały proces na podstawie świeżych wyników. Na przykład asystent konwersacyjny może potwierdzać wynik każdego podzadania przed zaplanowaniem kolejnego, dynamicznie dostosowując swoją sekwencję, aby zapewnić dużą szybkość reakcji i niezawodność działań.

Bez solidnej warstwy orkiestracji nawet najpotężniejsze funkcje mogą działać w sposób wzajemnie sprzeczny lub zupełnie stanąć w miejscu. Wzorce, architektury i najlepsze praktyki budowania odpornych, elastycznych mechanizmów orkiestracji omówimy szczegółowo w rozdziale 5.

Kompromisy projektowe

Projektowanie systemów agentowych wymaga znajdowania równowagi między wieloma kompromisami w celu zoptymalizowania efektywności, skalowalności, niezawodności i kosztów. Te kompromisy zmuszają programistów do podejmowania strategicznych decyzji, które mogą w znaczący sposób wpłynąć na to, jak agent radzi sobie w rzeczywistych warunkach. W tym podrozdziale omówię kluczowe dylematy towarzyszące tworzeniu skutecznych systemów agentowych i wskażę, jak najlepiej podchodzić do tych wyzwań.

Efektywność — kompromis między szybkością a dokładnością

Kluczowym kompromisem w projektowaniu agentów jest równoważenie szybkości i dokładności. Wysoka szybkość często umożliwia agentowi szybkie przetwarzanie informacji, podejmowanie decyzji i wykonywanie zadań, ale może się to odbywać kosztem precyzji. Z drugiej strony skupienie się na dokładności może spowolnić działanie agenta, szczególnie gdy pożądane są złożone modele lub techniki wymagające intensywnych obliczeń.

W środowiskach czasu rzeczywistego, takich jak pojazdy autonomiczne czy systemy transakcyjne, szybkie podejmowanie decyzji ma kluczowe znaczenie — czasami różnica kilku milisekund bywa krytyczna. W takich przypadkach nacisk na szybkość kosztem dokładności może być konieczny dla zapewnienia terminowej reakcji. Jednak zadania takie jak analiza prawna czy diagnostyka medyczna wymagają wysokiej precyzji, co sprawia, że akceptowalne jest poświęcenie części szybkości w celu zapewnienia wiarygodnych wyników.

Skuteczne może być również podejście hybrydowe, w którym agent początkowo dostarcza szybką, przybliżoną odpowiedź, a następnie przeprowadza bardziej dokładną analizę uzupełniającą. Takie podejście jest powszechne w systemach rekomendacyjnych czy diagnostycznych, w których szybka wstępna sugestia jest następnie weryfikowana i ulepszana dzięki dodatkowemu czasowi i danym.

Kup ksi k

Pole ksi k

Skalowalność — projektowanie skalowalnych systemów agentowych

Skalowalność stanowi kluczowe wyzwanie w nowoczesnych systemach agentowych, szczególnie tych, które w dużym stopniu opierają się na modelach uczenia głębokiego i na przetwarzaniu w czasie rzeczywistym. W miarę jak systemy agentów stają się coraz bardziej złożone pod względem objętości danych i równoczesności zadań, zarządzanie zasobami obliczeniowymi, zwłaszcza procesorami graficznymi, nabiera krytycznego znaczenia. Procesory GPU przyspieszają trenowanie i wnioskowanie dużych modeli AI, ale efektywne skalowanie wymaga starannego podejścia inżynierskiego w celu uniknięcia wąskich gardeł, niedostatecznego wykorzystania zasobów i rosnących kosztów operacyjnych. W tym punkcie omówię strategie skutecznego skalowania systemów agentowych poprzez optymalizację zasobów GPU i architektury.

Zasoby procesorów graficznych często stanowią najdroższy i najbardziej ograniczający czynnik w skalowaniu systemów agentowych, więc ich efektywne wykorzystanie staje się priorytetem. Właściwe zarządzanie zasobami umożliwia agentom radzenie sobie z rosnącym obciążeniem przy jednoczesnym minimalizowaniu opóźnień i kosztów obliczeniowych. Kluczową strategią skalowalności jest dynamiczna alokacja GPU, która polega na przydzielaniu zasobów procesorów graficznych w zależności od bieżącego zapotrzebowania. Zamiast statycznego przydzielania GPU agentom lub zadaniom, dynamiczna alokacja sprawia, że procesory graficzne są używane tylko wtedy, gdy jest to konieczne, co redukuje czas bezczynności i optymalizuje wykorzystanie.

Elastyczne przydzielanie zasobów GPU dodatkowo zwiększa efektywność poprzez wykorzystanie usług chmurowych lub lokalnych klastrów GPU, które automatycznie skalują zasoby w zależności od bieżącego obciążenia.

Kolejkowanie priorytetowe i inteligentne planowanie zadań dodają kolejną warstwę efektywności, zapewniając zadaniom o wysokim priorytecie natychmiastowy dostęp do GPU, podczas gdy mniej krytyczne zadania są kolejkowane w okresach szczytowego obciążenia.

W dużych systemach agentowych opóźnienia mogą stać się poważnym problemem, szczególnie gdy agenty muszą współdziałać w środowiskach czasu rzeczywistego lub niemal rzeczywistego. Optymalizacja pod kątem minimalnych opóźnień jest niezbędna, by agenty szybko reagowały i były zdolne do spełnienia wymagań wydajnościowych. Efektywne planowanie zadań GPU w systemach rozproszonych może zmniejszyć opóźnienia i zapewnić płynne działanie agentów pod dużym obciążeniem.

Jedną ze skutecznych strategii jest asynchroniczne wykonywanie zadań, które umożliwia przetwarzanie zadań GPU równolegle bez oczekiwania na zakończenie poprzednich zadań, co pozwala maksymalizować wykorzystanie zasobów GPU i redukować czas bezczynności między zadaniami.

Inną strategią jest dynamiczne równoważenie obciążenia między procesorami GPU, które zapobiega przekształceniu się pojedynczego GPU w wąskie gardło poprzez dystrybuowanie zadań do niedostatecznie wykorzystywanych zasobów. W przypadku systemów agentowych intensywnie wykorzystujących GPU, takich jak wykonywanie złożonych algorytmów wnioskowania, skuteczne skalowanie wymaga czegoś więcej niż zwykłe dodawanie procesorów graficznych — wymaga starannej optymalizacji w celu zapewnienia pełnego wykorzystania zasobów, żeby system efektywnie się skalował w miarę wzrostu wymagań.

Kup ksi k

Pole ksi k

Skalowanie horyzontalne polega na rozszerzaniu systemu poprzez dodawanie większej liczby węzłów GPU do obsługi rosnącego obciążenia. W konfiguracji klastrowej procesory GPU mogą współpracować w celu wykonywania zadań wymagających dużego nakładu pracy, takich jak wnioskowanie w czasie rzeczywistym czy trenowanie modeli.

W systemach agentowych o zmiennym obciążeniu wykorzystanie hybrydowego podejścia chmurowego może poprawić skalowalność poprzez połączenie lokalnych zasobów GPU z procesorami GPU działającymi w chmurze. W okresach szczytowego zapotrzebowania system może wykorzystać skalowanie tymczasowe, w którym zadania są przekazywane do GPU w chmurze, co zwiększa pojemność obliczeniową bez konieczności stałych inwestycji w infrastrukturę fizyczną. Po spadku zapotrzebowania zasoby te mogą zostać zwolnione, a to pozwala na utrzymanie efektywności kosztowej.

Wykorzystanie chmurowych instancji GPU w godzinach poza szczytem, kiedy zapotrzebowanie jest mniejsze, a ceny bardziej korzystne, może znacznie obniżyć koszty operacyjne przy zachowaniu elastyczności pozwalającej na skalowanie w górę w razie potrzeby.

Efektywne skalowanie systemów agentowych — szczególnie tych opartych na zasobach GPU — wymaga starannego zrównoważenia maksymalizacji wydajności GPU, minimalizacji opóźnień oraz zdolności systemu do radzenia sobie z dynamicznymi obciążeniami. Dzięki zastosowaniu strategii takich jak dynamiczna alokacja GPU, równoległość wieloprocesorowa, rozproszone wnioskowanie i hybrydowe infrastruktury chmurowe, systemy agentowe mogą się skalować, aby sprostać rosnącym wymaganiom przy zachowaniu wysokiej wydajności i efektywności kosztowej. W procesie tym kluczową rolę odgrywają narzędzia do zarządzania zasobami GPU, które zapewniają nadzór niezbędny do zagwarantowania płynnej skalowalności w miarę wzrostu złożoności i zakresu systemów agentowych.

Niezawodność — zapewnianie stabilnego i spójnego działania agentów

Niezawodność odnosi się do zdolności agenta do spójnego i dokładnego wykonywania zadań przez dłuższy czas. Niezawodny agent musi radzić sobie zarówno z przewidywalnymi, jak i nieprzewidywalnymi sytuacjami bez awarii, żeby użytkownicy i interesariusze darzyli go zaufaniem. Jednakże poprawa niezawodności często wiąże się z kompromisami dotyczącymi złożoności systemu, kosztów oraz czasu potrzebnego na rozwój.

Odporność na awarie

Jednym z kluczowych aspektów niezawodności jest zadbanie o to, aby agenty potrafiły radzić sobie z błędami lub nieoczekiwanymi zdarzeniami bez zawieszania się lub podejmowania nieprzewidywalnych działań. Może to wymagać wbudowania *odporności na awarie*, dzięki której agent będzie umiał wykrywać problemy (np. przerwy w połączeniu sieciowym, awarie sprzętu) i sprawnie wracać do normalnego stanu. Systemy odporne na awarie często wykorzystują *nadmiarowość* — powielanie krytycznych komponentów lub procesów, aby awaria jednej części systemu nie wpływała na jego ogólne działanie.

Kup ksi k Pole ksi k

Spójność i niezawodność

Aby agenty były niezawodne, muszą działać konsekwentnie w różnych scenariuszach, przy różnych danych wejściowych i w różnych środowiskach. Jest to szczególnie istotne w systemach, w których krytyczne znaczenie ma bezpieczeństwo, takich jak pojazdy autonomiczne czy urządzenia medyczne, ponieważ błędy w nich mogą mieć poważne konsekwencje. Programiści muszą zadbać o to, by agent działał dobrze nie tylko w idealnych warunkach, ale także w przypadkach granicznych, podczas testów obciążeniowych i przy rzeczywistych ograniczeniach środowiskowych. Do osiągnięcia niezawodności niezbędne są:

Rozległe testy

Agenty powinny przechodzić rygorystyczne testy, w tym testy jednostkowe, testy integracyjne oraz symulacje rzeczywistych scenariuszy. Testy powinny obejmować przypadki graniczne, nieoczekiwane dane wejściowe oraz warunki adwersaryjne, aby uzyskać pewność, że agent poradzi sobie w różnorodnych środowiskach.

Monitorowanie i pętle informacji zwrotnej

Niezawodne agenty wymagają ciągłego monitorowania w środowisku produkcyjnym w celu wykrywania anomalii i dostosowywania działania w reakcji na zmieniające się warunki. Pętle informacji zwrotnej umożliwiają agentom uczenie się ze swojego środowiska i stopniowe poprawianie efektywności, przez co zwiększają ich odporność.

Koszty — równowaga między wydajnością a wydatkami

Koszty to często pomijany, lecz kluczowy czynnik w projektowaniu systemów agentowych. Nakłady związane z opracowaniem, wdrożeniem i utrzymaniem agenta muszą zostać zestawione z oczekiwanymi korzyściami i zwrotem z inwestycji. Kwestie kosztowe wpływają na decyzje dotyczące złożoności modelu, infrastruktury oraz skalowalności.

Koszty rozwojowe

Tworzenie zaawansowanych agentów może być kosztowne, szczególnie gdy korzysta się z nowoczesnych modeli uczenia maszynowego, które wymagają dużych zbiorów danych, specjalistycznej wiedzy oraz znacznych zasobów obliczeniowych do trenowania. Ponadto koszty rozwojowe podnosi potrzeba iteracyjnego projektowania, testowania i optymalizacji.

Złożone agenty często wymagają zespołu specjalistów o różnych umiejętnościach, w tym naukowców, inżynierów uczenia maszynowego oraz ekspertów dziedzinowych. By zbudować niezawodny i skalowalny system agentowy, potrzeba natomiast rozbudowanej infrastruktury testowej, często obejmującej środowiska symulacyjne oraz inwestycje w narzędzia i platformy testowe.

Koszty operacyjne

Koszty operacyjne związane z używaniem agentów często okazują się znaczące, szczególnie w przypadku systemów wymagających dużej mocy obliczeniowej, na przykład zajmujących się podejmowaniem decyzji w czasie rzeczywistym lub ciągłym przetwarzaniem danych. Głównym

Kup ksi k Pole ksi k

składnikiem tych wydatków jest koszt zasobów obliczeniowych, ponieważ agenty wykonujące modele uczenia głębokiego lub złożone algorytmy często polegają na kosztownym sprzęcie, takim jak procesory GPU czy usługi chmurowe.

Ponadto agenty przetwarzające ogromne ilości danych lub utrzymujące rozległą pamięć generują wyższe koszty związane z przechowywaniem danych i łącznością sieciową. Regularna konserwacja i aktualizacje, w tym poprawki błędów i ulepszenia systemu, dodatkowo zwiększają wydatki operacyjne, ponieważ potrzebne są zasoby zapewniające długoterminową niezawodność i wydajność systemu.

Koszt a wartość

Ostatecznie koszt systemu agentowego musi być uzasadniony wartością, jaką ten system dostarcza. W niektórych przypadkach rozsądne może być priorytetowe używanie tańszych, prostszych agentów do mniej istotnych zadań przy jednoczesnym inwestowaniu znacznych środków w bardziej zaawansowane agenty do zastosowań krytycznych. Decyzje dotyczące kosztów trzeba podejmować w kontekście ogólnych celów systemu i jego przewidywanego czasu eksploatacji. Oto niektóre strategie optymalizacji:

Lekkie modele

Stosowanie prostszych, bardziej efektywnych modeli tam, gdzie jest to właściwe, może pomóc w zmniejszeniu zarówno kosztów rozwoju, jak i kosztów operacyjnych. Na przykład jeśli w konkretnym zadaniu system oparty na regułach może osiągnąć podobne rezultaty jak model uczenia głębokiego, prostsze podejście będzie często bardziej opłacalne.

Zasoby chmurowe

Wykorzystanie zasobów chmurowych może zmniejszyć początkowe koszty infrastruktury, gdyż pozwala na wprowadzenie skalowalnego modelu płatności za rzeczywiste użycie.

Modele i narzędzia open source

Korzystanie z bibliotek i platform uczenia maszynowego o otwartym kodzie źródłowym może pomóc zminimalizować koszty rozwoju oprogramowania, a jednocześnie zapewnić wysoką jakość agentów.

Projektowanie systemów agentowych wymaga uwzględnienia kilku kluczowych kompromisów. Priorytetowe traktowanie wydajności może wymagać poświęcenia części dokładności, podczas gdy skalowanie do architektury wieloagentowej utrudnia zachowanie koordynacji i spójności. Zapewnienie niezawodności wymaga rygorystycznego testowania i monitorowania, ale może wydłużyć czas tworzenia oprogramowania i zwiększyć jego złożoność. Wreszcie kwestie kosztów trzeba rozpatrzyć zarówno z perspektywy rozwojowej, jak i operacyjnej, aby system dostarczał wartość w ramach ograniczonego budżetu. W następnym podrozdziale przeanalizuję niektóre z najczęściej stosowanych wzorców projektowych używanych do budowania efektywnych systemów agentowych.

Kup ksi k

Pole ksi k

Architekturalne wzorce projektowe

Projekt architektoniczny systemu agentowego określa strukturę agentów, ich interakcje ze środowiskiem oraz sposób wykonywania zadań. Wybór architektury wpływa na skalowalność, łatwość konserwowania i elastyczność systemu. W tym podrozdziale omówię dwa popularne wzorce projektowe systemów agentowych — architektury jednoagentowe i wieloagentowe — oraz zbadam ich zalety, wady oraz potencjalne zastosowania. Zagadnienia te zostaną opisane szczegółowo w rozdziale 8.

Architektury jednoagentowe

Jednym z najprostszych i najbardziej przejrzystych rozwiązań projektowych jest architektura jednoagentowa, w której pojedynczy agent odpowiada za wykonywanie wszystkich zadań w systemie. Agent ten wchodzi w bezpośrednie interakcje ze swoim środowiskiem i samodzielnie zajmuje się podejmowaniem decyzji, planowaniem oraz realizacją zadań, bez polegania na innych agentach.

Architektura ta nadaje się idealnie do dobrze zdefiniowanych, wąsko zakrojonych zadań i najlepiej sprawdza się w przypadku obciążeń, z którymi może poradzić sobie pojedyncza jednostka. Prostota systemów jednoagentowych czyni je łatwymi w projektowaniu, tworzeniu i wdrażaniu, ponieważ pozwala uniknąć złożoności związanych z koordynacją, komunikacją i synchronizacją między wieloma komponentami. Architektury jednoagentowe sprawdzają się doskonale w zadaniach o wąskim zakresie, które nie wymagają współpracy ani rozproszonych działań, takich jak proste czatboty obsługujące podstawowe zapytania klientów (np. często zadawane pytania czy śledzenie zamówień) oraz automatyzacja konkretnych zadań związanych z wprowadzaniem danych lub zarządzaniem plikami.

Konfiguracje jednoagentowe dobrze funkcjonują w środowiskach, w których dziedzina problemu jest dobrze zdefiniowana, zadania są proste i nie ma znaczącej potrzeby skalowania. Sprawdzają się więc w przypadku czatbotów obsługi klienta, asystentów ogólnego przeznaczenia oraz agentów generujących kod. Architektury jednoagentowe i wieloagentowe omówię znacznie szerzej w rozdziale 8.

Architektury wieloagentowe — współpraca, przetwarzanie równoległe i koordynacja

W architekturach wieloagentowych wiele agentów współpracuje ze sobą w celu osiągnięcia wspólnego celu. Agenty te mogą działać niezależnie, równolegle lub w sposób skoordynowany, w zależności od charakteru wykonywanych zadań. Systemy wieloagentowe są często wykorzystywane w złożonych środowiskach, w których różne aspekty zadania muszą być realizowane przez wyspecjalizowane jednostki lub w których przetwarzanie równoległe może poprawić wydajność i skalowalność. Przynoszą one wiele korzyści:

Kup ksi k Pole ksi k

Współpraca i specjalizacja

Każdy agent w systemie wieloagentowym może być zaprojektowany tak, aby specjalizował się w określonych zadaniach lub dziedzinach. Na przykład jeden agent może skupiać się na zbieraniu danych, podczas gdy drugi przetwarza te dane, a trzeci zarządza interakcjami z użytkownikami. Taki podział pracy umożliwia systemowi radzenie sobie ze złożonymi zadaniami znacznie efektywniej, niż mógłby to robić pojedynczy agent.

Paralelizacja

Architektury wieloagentowe mogą wykonywać wiele zadań jednocześnie. Na przykład agenty w systemie logistycznym mogą równocześnie planować różne trasy dostaw, co pozwala skrócić całkowity czas przetwarzania i poprawić wydajność.

Większa skalowalność

W miarę rozwoju systemu można wprowadzać dodatkowe agenty do obsługi większej liczby zadań lub do równoważenia obciążenia. Dzięki temu systemy wieloagentowe są wysoce skalowalne i zdolne do zarządzania większymi, bardziej złożonymi środowiskami.

Redundancja i odporność

Ponieważ wiele agentów działa niezależnie, awaria jednego agenta niekoniecznie zagraża całemu systemowi. Pozostałe agenty mogą kontynuować pracę, a nawet przejąć obowiązki uszkodzonego agenta, co poprawia ogólną niezawodność systemu.

Pomimo tych zalet systemy wieloagentowe wiążą się również z istotnymi wyzwaniami, do których należą:

Koordynacja i komunikacja

Zarządzanie komunikacją między agentami może być złożone. Agenty muszą efektywnie wymieniać informacje i koordynować swoje działania, aby uniknąć powielania pracy, sprzecznych działań czy rywalizacji o zasoby. Bez odpowiedniej koordynacji systemy wieloagentowe stają się chaotyczne i nieefektywne.

Większa złożoność

Systemy wieloagentowe są potężne, ale również bardziej wymagające w projektowaniu, rozwoju i utrzymaniu. Potrzeba stosowania protokołów komunikacyjnych, strategii koordynacji i mechanizmów synchronizacji wymusza dodanie kolejnych warstw złożoności do architektury systemu.

Niższa efektywność

Choć nie jest to regułą, systemy wieloagentowe często napotykają problemy z efektywnością z powodu zwiększonego zużycia tokenów podczas realizacji zadań. Ponieważ agenty muszą często się komunikować, dzielić kontekstem i koordynować swoje działania, zużywają więcej mocy obliczeniowej i zasobów w porównaniu z systemami jednoagentowymi. To zwiększone wykorzystanie tokenów nie tylko prowadzi do wyższych kosztów obliczeniowych, lecz także może spowalniać wykonywanie zadań, jeśli komunikacja i koordynacja nie są odpowiednio zoptymalizowane. W konsekwencji, choć systemy wieloagentowe oferują solidne rozwiązania dla złożonych zadań, problemy z efektywnością oznaczają, że kluczowe znaczenie ma staranne zarządzanie zasobami.

Kup ksi k Pole ksi k

Architektury wieloagentowe sprawdzają się dobrze w środowiskach, w których zadania są złożone, rozproszone lub wymagają specjalizacji poszczególnych komponentów. W takich systemach wiele agentów ma wkład w rozwiązywanie skomplikowanych, rozproszonych problemów, na przykład w systemach handlu finansowego, dochodzeniach dotyczących cyberbezpieczeństwa czy platformach do współpracy w badaniach nad sztuczną inteligencją.

Systemy jednoagentowe oferują prostotę i nadają się idealnie do dobrze zdefiniowanych zadań. Systemy wieloagentowe zapewniają współpracę, równoległość i skalowalność, co czyni je odpowiednimi dla złożonych środowisk. Wybór właściwej architektury zależy od złożoności zadania, potrzeby skalowalności i przewidywanego czasu użytkowania systemu. W następnym podrozdziale omówimy niektóre zasady, których powinieneś przestrzegać, aby budowane przez Ciebie systemy agentowe osiągały optymalne wyniki.

Najlepsze praktyki

Projektowanie systemów agentowych to znacznie więcej niż budowanie agentów z odpowiednimi modelami, umiejętnościami i architekturą. Aby zapewnić optymalną wydajność takich systemów w rzeczywistych warunkach oraz ich ciągłą ewolucję w ślad za zmianami w środowisku, kluczowe jest przestrzeganie sprawdzonych praktyk przez cały cykl rozwoju. W niniejszym rozdziale opiszę trzy fundamentalne zasady — **projektowanie iteracyjne**, **strategię ewaluacji** oraz **testowanie w warunkach rzeczywistych** — które przyczyniają się do tworzenia adaptowalnych, wydajnych i niezawodnych systemów agentowych.

Projektowanie iteracyjne

Projektowanie iteracyjne to podejście do tworzenia agentów, które podkreśla znaczenie budowania systemów w sposób przyrostowy przy jednoczesnym ciągłym uwzględnianiu informacji zwrotnych. Zamiast dążyć do doskonałego rozwiązania już w pierwszej wersji, projektowanie iteracyjne skupia się na tworzeniu małych, funkcjonalnych prototypów, które można oceniać, ulepszać i dopracowywać w kolejnych cyklach. Ten proces umożliwia szybkie wykrywanie błędów, natychmiastowe korekty kierunku oraz ciągłe doskonalenie systemu, co niesie ze sobą liczne korzyści:

Wczesne wykrywanie problemów

Dzięki udostępnianiu wczesnych prototypów można zidentyfikować wady projektowe lub wąskie gardła, zanim głęboko zakorzenią się w systemie. Pozwala to na szybkie rozwiązywanie problemów, zmniejszenie długoterminowych kosztów rozwoju oraz uniknięcie poważnych przeróbek.

Projektowanie zorientowane na użytkownika

Projektowanie iteracyjne zachęca do częstego zbierania opinii od interesariuszy, użytkowników końcowych i innych programistów. Dzięki tym informacjom zwrotnym system agenta pozostaje zgodny z potrzebami i oczekiwaniami użytkowników. W miarę testowania agentów w rzeczywistych scenariuszach iteracyjne ulepszenia pozwalają dostrajać ich zachowania i reakcje, aby lepiej służyły użytkownikom.

Kup ksi k Pole ksi k

Skalowalność

Rozpoczęcie od produktu o minimalnej koniecznej funkcjonalności (ang. *minimal viable product*, MVP) lub podstawowego agenta pozwala na rozwijanie i modyfikowanie systemu w możliwych do opanowania krokach. W miarę dojrzewania systemu nowe funkcje i możliwości można wprowadzać stopniowo, dokładnie testując każdy dodatek przed pełnym wdrożeniem.

Aby skutecznie zastosować projektowanie iteracyjne, zespoły programistyczne powinny:

Szybko tworzyć prototypy

Skupić się najpierw na budowaniu podstawowej funkcjonalności. Na tym etapie nie należy dążyć do perfekcji — trzeba zbudować coś, co działa i dostarcza wartość, nawet jeśli jest to rozwiązanie podstawowe.

Testować i zbierać opinie

Po każdej iteracji należy zbierać opinie od użytkowników, programistów i innych interesariuszy. Te informacje zwrotne powinny służyć jako przewodnik po potencjalnych ulepszeniach i priorytetach w kolejnej iteracji.

Dopracowywać i powtarzać

Na podstawie opinii i danych o wydajności należy wprowadzać niezbędne zmiany i dopracowywać system w kolejnej iteracji. Cykl ten należy kontynuować tak długo, aż system agenta osiągnie swoje cele w zakresie wydajności, użyteczności i skalowalności.

Skuteczne projektowanie iteracyjne polega na szybkim tworzeniu funkcjonalnych prototypów, zbieraniu opinii po każdej iteracji oraz ciągłym doskonaleniu systemu na podstawie zdobytych doświadczeń. Pozwala to osiągnąć cele związane z wydajnością i użytecznością.

Strategia ewaluacji

Ocena wydajności i niezawodności systemów agentowych stanowi kluczowy element procesu rozwoju. Solidna ewaluacja gwarantuje, że agenty będą radzić sobie z rzeczywistymi scenariuszami, działać w różnych warunkach i spełniać oczekiwania dotyczące wydajności. Wymaga to systematycznego podejścia do testowania i walidacji agentów w różnych wymiarach, w tym dokładności, efektywności, odporności i skalowalności. W tym punkcie opiszę kluczowe strategie tworzenia kompleksowych ram ewaluacyjnych dla systemów agentowych. Zagadnienia pomiaru i walidacji przedstawię znacznie szerzej w rozdziale 9.

Solidny proces ewaluacji polega na opracowaniu kompleksowych ram testowych, które obejmują wszystkie aspekty funkcjonalności agenta. Takie ramy umożliwiają gruntowne przetestowanie agenta w różnorodnych scenariuszach, zarówno przewidywalnych, jak i nieoczekiwanych.

Testowanie funkcjonalne koncentruje się na sprawdzeniu, czy agent prawidłowo wykonuje swoje podstawowe zadania. Każdą umiejętność czy moduł agenta należy testować indywidualnie, aby upewnić się, że zachowuje się zgodnie z oczekiwaniami przy różnych danych wejściowych i scenariuszach. Kluczowe obszary uwagi to m.in.:

Kup ksi k Pole ksi k

Poprawność

Zapewnienie, że agent konsekwentnie dostarcza dokładne i oczekiwane wyniki zgodnie ze swoim projektem.

Testowanie przypadków brzegowych

Ocena sposobu, w jaki agent radzi sobie z nietypowymi przypadkami i ekstremalnymi danymi wejściowymi, takimi jak bardzo duże zbiory danych, nietypowe zapytania czy niejednoznaczne instrukcje.

Miary specyficzne dla zadań

W przypadku agentów obsługujących zadania z konkretnych dziedzin (np. analiza prawna, diagnostyka medyczna) — zapewnienie, że system spełnia wymagania dotyczące dokładności i zgodności z przepisami w danej dziedzinie.

W przypadku systemów agentowych, szczególnie tych opartych na modelach uczenia maszynowego, kluczowe jest sprawdzenie zdolności agenta do generalizacji poza konkretne scenariusze, na których był trenowany. Gwarantuje to, że agent poradzi sobie z nowymi, nieznanymi sytuacjami, zachowując przy tym dokładność i niezawodność.

Agenty często napotykają zadania wykraczające poza pierwotny obszar treningu. Solidna ewaluacja powinna sprawdzać, czy agent potrafi adaptować się do nowych zadań bez konieczności ponownego rozległego trenowania. Jest to szczególnie istotne w przypadku agentów uniwersalnych lub zaprojektowanych do działania w dynamicznych środowiskach.

Kluczowym czynnikiem decydującym o sukcesie systemów agentowych są wrażenia użytkownika. Ważne jest, aby oceniać nie tylko techniczną wydajność agenta, lecz także to, jak dobrze spełnia oczekiwania użytkowników w rzeczywistych zastosowaniach.

Zbieranie opinii od rzeczywistych użytkowników dostarcza cennych informacji o tym, jak dobrze agent sprawdza się w praktyce. Te opinie pomagają poprawiać skuteczność agenta oraz satysfakcję użytkowników i mogą obejmować następujące elementy:

Wskaźniki zadowolenia użytkowników

Wykorzystuj wskaźniki takie jak NPS (Net Promoter Score) lub CSAT (Customer Satisfaction), aby ocenić, jak użytkownicy postrzegają swoje interakcje z agentem.

Współczynniki ukończenia zadań

Mierz, jak często użytkownicy pomyślnie wykonują zadania za pomocą agenta. Niskie współczynniki ukończenia mogą wskazywać na chaotyczne działanie lub nieefektywność projektu agenta.

Sygnały jawne

Zapewnij użytkownikom możliwość przekazywania opinii w takich formach jak kciuk w górę i w dół, oceny gwiazdkowe oraz możliwość akceptowania, odrzucania lub modyfikowania wygenerowanych wyników w zależności od kontekstu. Sygnały te mogą dostarczyć cennych informacji.

Kup ksi k Pole ksi k

Sygnały niejawne

Analizuj interakcje między użytkownikiem a agentem, aby zidentyfikować typowe problemy, takie jak błędne interpretacje, opóźnienia lub niewłaściwe odpowiedzi. W poszukiwaniu obszarów wymagających poprawy można przeanalizować dzienniki interakcji.

W niektórych przypadkach trzeba zaangażować ludzkich ekspertów w proces ewaluacji, aby ocenić dokładność podejmowania decyzji przez agenta. Walidacja z człowiekiem w pętli łączy automatyczną ewaluację z ludzkim osądem, co pozwala ocenić skuteczność agenta zgodnie ze standardami rzeczywistego świata. Gdy to możliwe, eksperci ludzcy powinni przejrzeć próbkę wyników agenta, aby zweryfikować poprawność, zgodność etyczną i zgodność z najlepszymi praktykami. Przeglądy te można następnie wykorzystać do kalibracji i poprawy automatycznych ewaluacji.

Agentów należy oceniać w środowiskach, które ściśle symulują ich rzeczywiste zastosowania. Pomaga to zapewnić niezawodne działanie systemu poza kontrolowanymi warunkami rozwojowymi. Oceniaj agenta w całym spektrum jego środowiska operacyjnego, od pobierania i przetwarzania danych po wykonywanie zadań i generowanie wyników. Kompleksowe testy dają pewność, że agent operuje zgodnie z oczekiwaniami w wielu systemach, na różnych źródłach danych i platformach.

Testowanie w warunkach rzeczywistych

Chociaż tworzenie agentów w kontrolowanym środowisku programistycznym ma kluczowe znaczenie dla wstępnych testów, równie ważne jest sprawdzenie ich działania w rzeczywistych warunkach, aby upewnić się, że będą funkcjonować zgodnie z oczekiwaniami podczas interakcji z prawdziwymi użytkownikami lub środowiskami. Testowanie w warunkach rzeczywistych polega na wdrożeniu agentów w faktycznych środowiskach produkcyjnych i obserwowaniu ich zachowania w praktycznych sytuacjach. Ten etap testowania umożliwia programistom wykrycie problemów, które mogły się nie ujawnić podczas wcześniejszych faz rozwojowych, oraz ocenę odporności, niezawodności i wpływu agenta na użytkowników.

Testowanie w warunkach rzeczywistych jest niezbędne, aby zapewnić, że agenty poradzą sobie z nieprzewidywalnością i złożonością środowiska produkcyjnego. Takie podejście — w przeciwieństwie do testów kontrolowanych — ujawnia przypadki brzegowe, nieoczekiwane dane wejściowe od użytkowników oraz wydajność przy wysokim obciążeniu, co pomaga programistom doskonalić agenta w celu zapewnienia solidnego i niezawodnego działania:

Kontakt ze złożonością rzeczywistego świata

W kontrolowanych środowiskach agenty działają z przewidywalnymi danymi wejściowymi i odpowiedziami. Jednak rzeczywiste środowiska są dynamiczne i nieprzewidywalne, z różnorodnymi użytkownikami, przypadkami brzegowymi i nieprzewidzianymi wyzwaniami. Testowanie w takich środowiskach gwarantuje, że agent poradzi sobie ze złożonością i zmiennością praktycznych scenariuszy.

Kup ksi k Pole ksi k

Odkrywanie przypadków brzegowych

Interakcje w rzeczywistych warunkach często ujawniają przypadki brzegowe, które mogły nie zostać uwzględnione na etapie projektowania lub testowania. Na przykład czatbot testowany za pomocą przygotowanych zapytań może dobrze działać w fazie rozwojowej, ale gdy zostanie wystawiony na kontakt z prawdziwymi użytkownikami, może mieć trudności z nieoczekiwanymi danymi wejściowymi, niejasnymi pytaniami lub wariantami języka naturalnego.

Ocena wydajności pod obciążeniem

Testowanie w warunkach rzeczywistych pozwala programistom obserwować, jak agent radzi sobie przy wysokim obciążeniu lub zwiększonym zapotrzebowaniu ze strony użytkowników. Jest to szczególnie istotne dla agentów działających w środowiskach o zmiennym natężeniu ruchu, takich jak boty do obsługi klienta czy systemy rekomendacji w e-commerce.

Testowanie w rzeczywistych warunkach pozwala ocenić gotowość agenta do wdrożenia. Proces ten obejmuje wieloetapowe wdrożenie, ciągłe monitorowanie kluczowych wskaźników, zbieranie opinii użytkowników oraz iteracyjne udoskonalanie agenta w celu optymalizacji jego możliwości i użyteczności:

Wdrażanie wieloetapowe

Wdrażaj agenta etapami. Zaczynaj od testów na małą skalę w ograniczonym środowisku, a następnie stopniowo przechodź do pełnego wdrożenia. Takie podejście pomaga identyfikować i rozwiązywać problemy krok po kroku bez przeciążania systemu oraz użytkowników.

Monitorowanie zachowania agenta

Używaj narzędzi monitorujących do śledzenia zachowania agenta, jego odpowiedzi i wskaźników wydajności podczas testowania w warunkach rzeczywistych. Monitorowanie powinno skupiać się na kluczowych wskaźnikach wydajności (KPI), takich jak czas odpowiedzi, dokładność, zadowolenie użytkowników i stabilność systemu.

Zbieranie opinii użytkowników

Angażuj użytkowników podczas testów w rzeczywistych warunkach, aby zebrać opinie na temat ich wrażeń z interakcji z agentem. Opinie użytkowników są nieocenione w identyfikowaniu luk, poprawianiu użyteczności i dbaniu o to, by agent spełniał rzeczywiste potrzeby.

Iteracja bazująca na spostrzeżeniach

Testowanie w rzeczywistych warunkach dostarcza cennych spostrzeżeń, które powinny zostać uwzględnione w cyklu rozwojowym. Wykorzystaj te informacje do udoskonalania agenta, poprawy jego możliwości i optymalizacji wydajności w kolejnych iteracjach.

Przestrzeganie najlepszych praktyk, takich jak projektowanie iteracyjne, zwinne wytwarzanie oprogramowania oraz testowanie w rzeczywistych warunkach, ma kluczowe znaczenie dla budowania systemów agentowych, które są adaptowalne, skalowalne i odporne. Praktyki te pozwalają projektować agenty pod kątem elastyczności, dokładnie testować je w rzeczywistych warunkach i stale udoskonalać, aby mogły sprostać zmieniającym się potrzebom użytkowników

Kup ksi k Pole ksi k

i wyzwaniom środowiskowym. Dzięki włączeniu tych podejść do cyklu rozwojowego programiści mogą tworzyć bardziej niezawodne, wydajne i skuteczne systemy agentowe zdolne do skutecznego działania w dynamicznych środowiskach.

Podsumowanie

Nie potrzebujesz 30-stronicowego planu, żeby zacząć budować dobry system agentowy — ale odrobina przezorności bardzo się przydaje. Jak widzieliśmy na przykładzie naszego agenta e-commerce, wybranie konkretnego fragmentu funkcjonalności — takiego jak anulowanie zamówień — pozwala zbudować coś małego, możliwego do przetestowania i od razu użytecznego. Określ, jak ma wyglądać sukces, unikaj niejasnych lub zbyt ambitnych celów i skup się na szybkim dostarczaniu konkretnej wartości.

Skuteczne systemy agentowe to coś więcej niż suma ich części. Wymagają one solidnej architektury, zdyscyplinowanego podejścia do programowania i sprawnych pętli informacji zwrotnej. Wybór odpowiedniego wzorca strukturalnego przygotowuje grunt pod skalowalność i odporność systemu, podczas gdy iteracyjny rozwój i rzetelna ewaluacja umożliwiają ulepszanie agentów z biegiem czasu. Sprawdzone praktyki, takie jak stopniowe wdrażanie i testowanie w rzeczywistych warunkach, przekształcają obiecujące prototypy — jak nasz prosty agent do anulowania zamówień — w niezawodne systemy, którym można zaufać w środowisku produkcyjnym.

W rozdziale 3. przeniesiemy uwagę na ludzką stronę równania i porozmawiamy o projektowaniu agentów w taki sposób, żeby działały w sposób czytelny i intuicyjny dla użytkowników. Ostatecznie bez względu na to, jak potężna jest architektura systemu, jego sukces zależy od tego, czy sprawdzi się on w ludzkich rękach.

Kup ksi k

Pole ksi k

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

Projektuj agenty AI, które analizują, uczą się i działają dla biznesu

Systemy oparte na agentach AI rewolucjonizują sposób, w jaki organizacje rozwiązują złożone problemy biznesowe. Generatywna sztuczna inteligencja przyspiesza drogę od koncepcji przez prototyp aż do gotowego rozwiązania, a agenty AI, łączące narzędzia, wiedzę, pamięć i uczenie się z zaawansowanymi modelami podstawowymi — umożliwiają sekwencyjne wywoływanie modeli do realizacji niejednoznacznych zadań. Od agentów-koderów, przez agentów-badaczy, po agentów-analityków — wszędzie widać, że mogą one znacząco przyspieszyć pracę zespołów. Jednak ich wdrożenie pozostaje wyzwaniem wymagającym intensywnego planowania.

Książka stanowi praktyczny przewodnik po projektowaniu i wdrażaniu systemów jedno- i wieloagentowych. Autor wyjaśnia szczegółowo komponenty agentów AI — od wyboru modelu, przez narzędzia i pamięć, po orkiestrację i planowanie. Przedstawia kompromisy projektowe, wzorce architektoniczne, a także najlepsze praktyki w zakresie skalowalności, współpracy między agentami i projektowania interfejsów użytkownika. Omawia również kluczowe aspekty pomiarów, walidacji, monitorowania produkcyjnego i bezpieczeństwa systemów agentowych.

W książce:

- Szczególne cechy agentów AI opartych na modelach podstawowych
- Kluczowe komponenty i zasady projektowe agentów AI
- Kompromisy projektowe i wdrażanie efektywnych systemów wieloagentowych
- Projektowanie i wdrażanie dostosowanych rozwiązań AI, zwiększających efektywność i innowacyjność

Michael Albada jest inżynierem uczenia maszynowego z dziewięcioletnim doświadczeniem we wdrażaniu rozwiązań ML w dużych firmach technologicznych: Uber, ServiceNow i Microsoft. Specjalizuje się w dużych modelach językowych, systemach wieloagentowych i cyberbezpieczeństwie. Uzyskał tytuły naukowe związane z uczeniem maszynowym na uniwersytetach Stanforda i Cambridge, a także w Georgia Tech.

Najlepsze jednotomowe wprowadzenie do budowania agentowych systemów AI — możesz przeczytać setki prac naukowych albo tę jedną książkę!

Arun Rao, były pracownik działu GenAI w firmie Meta, profesor nadzwyczajny UCLA

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 978-83-289-3682-9

Cena: 99,00 zł