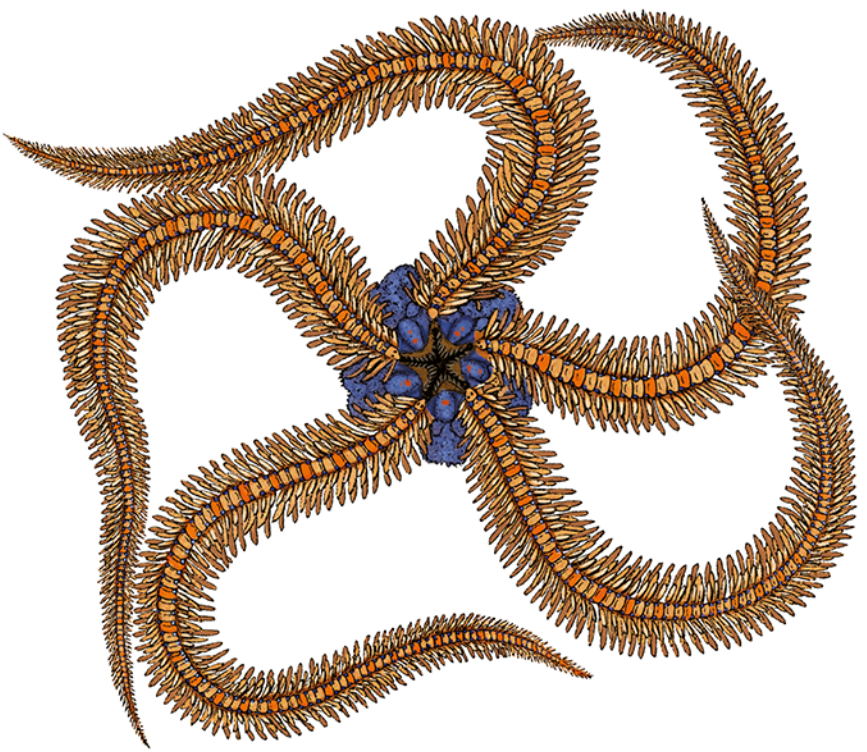
O'REILLY®

Tworzenie aplikacji z wykorzystaniem GPT-4 i ChatGPT

Buduj inteligentne chatboty, generatory treści i fascynujące projekty



Olivier Caelen
Marie-Alice Blete

Helion

Tytuł oryginału: Developing Apps with GPT-4 and ChatGPT: Build Intelligent Chatbots, Content Generators, and More

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-289-1044-7

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

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

Printed in Poland.

Spis treści

ROZDZIAŁ 2.

Szczegółowe informacje o interfejsach API modeli GPT-4 i ChatGPT

W tym rozdziale dokładnie przyjrzymy się interfejsom API modeli GPT-4 i ChatGPT. Celem jest dogłębne przedstawienie zasady ich funkcjonowania, abyś mógł je stosować w aplikacjach w języku Python. Po przeczytaniu tego rozdziału będziesz potrafił wykorzystywać potężne możliwości tych interfejsów we własnych projektach programistycznych.

Najpierw poznasz platformę OpenAI Playground, dzięki której dokładniej zrozumiesz modele, zanim zaczniesz pisać kod. Następnie przyjrzymy się bibliotece OpenAI. Napiszesz prosty kod logujący się do usługi i wyświetlający napis „Witaj, świecie!". Poznasz proces tworzenia i wysyłania zapytań do interfejsów API. Dowiesz się również, jak przetwarzać odpowiedzi i interpretować zwracane dane. Ponadto zapoznasz się z dobrymi praktykami w zakresie bezpieczeństwa i kontroli kosztów.

Podczas lektury będziesz zdobywał praktyczną, programistyczną wiedzę, która bardzo Ci się przyda w przygodzie z językiem Python oraz modelami GPT-4 i ChatGPT. Wszystkie opisane przykłady możesz pobrać pod adresem *https://ftp.helion.pl/przyklady/twapwy.zip*.

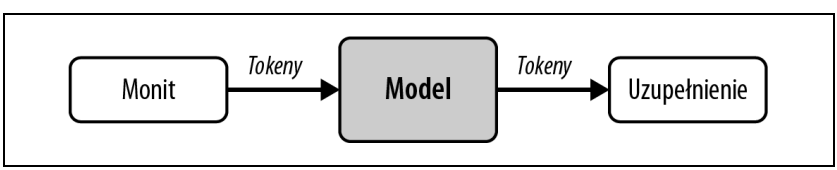
Zanim zaczniesz, zapoznaj się z zasadami korzystania z usług OpenAI (*https://openai.com/policies/usage-policies*). Jeżeli nie masz jeszcze konta, utwórz je na głównej stronie (*https://openai.com*). Przeczytaj również przepisy prawne na stronie *Terms & policies* (Warunki i zasady, *https://openai.com/policies*). Podczas korzystania z interfejsów API i bibliotek OpenAI niezbędna będzie znajomość pojęć opisanych w rozdziale 1.

Podstawowe pojęcia

Laboratorium OpenAI oferuje kilka modeli przeznaczonych do różnych zastosowań. Każdy ma własny cennik. Na kolejnych stronach znajdziesz szczegółowe porównanie dostępnych modeli i wskazówki, który z nich wybrać. Pamiętaj, że od przeznaczenia modelu (prowadzenia konwersacji, uzupełniania lub edycji tekstu) zależy sposób korzystania z jego interfejsu API. Na przykład modele ChatGPT i GPT-4 bazują na konwersacji i wykorzystują właściwy dla niej punkt końcowy.

W rozdziale 1. poznałeś pojęcie monitu. Monit nie jest atrybutem interfejsu OpenAI API, ale stanowi punkt wejścia do każdego modelu LLM. Mówiąc najprościej, jest to tekst wejściowy, wysyłany do modelu. Jest to komunikat dla modelu, jakie konkretnie zadanie ma wykonać. W przypadku modeli ChatGPT i GPT-4 monity tworzą konwersację, czyli listę komunikatów wejściowych i wyjściowych. W tym rozdziale dokładnie poznasz format tego rodzaju monitu.

W rozdziale 1. opisany został również token, czyli słowo lub jego część. W języku angielskim 75 słowom odpowiada około 100 tokenów. Opłata za zapytanie wysłane do interfejsu API modelu OpenAI zależy od liczby użytych tokenów, a więc od długości tekstów wejściowego i wyjściowego. Więcej szczegółów na temat zarządzania tokenami wejściowymi i wyjściowymi oraz kontrolowania ich liczby znajdziesz w podrozdziałach „Korzystanie z modeli ChatGPT i GPT-4” i „Korzystanie z innych modeli uzupełniających tekst”. Rysunek 2.1 przedstawia podsumowanie powyższych pojęć.



Rysunek 2.1. Podstawowe pojęcia w interfejsach OpenAI API

Po omówieniu pojęć przejdźmy do szczegółów modeli.

Dostępne interfejsy API modeli OpenAI

Za pomocą interfejsów API można korzystać z modeli opracowanych przez laboratorium OpenAI (*https://platform.openai.com/docs/models*). Modele są usługami, do których można wysyłać zapytania bezpośrednio lub za pomocą dostępnych bibliotek. Laboratorium OpenAI uruchomiło swoje modele na zewnętrznych serwerach i programiści mogą do nich wysyłać zapytania.

Każdy model oferuje inne funkcje i ma inną cenę. W tym podrozdziale przyjrzymy się modelom LLM. Pamiętaj, że są one chronione prawami autorskimi, więc nie można ich dostosowywać do własnych potrzeb, bezpośrednio modyfikując kody. Jak się jednak przekonasz, niektóre modele można dopasowywać do konkretnych danych, używając interfejsów OpenAI API.

Niektóre starsze modele nie są zastrzeżone. Na przykład model GPT-2 można pobrać z serwisu Hugging Face (*https://hugging-face.co/gpt2*) lub GitHub (*https://github.com/openai/gpt-2*). Nie można jednak z niego korzystać za pomocą interfejsu API.

Laboratorium OpenAI nieustannie aktualizuje wiele swoich modeli, więc nie sposób podać w książce ich pełnej, aktualnej listy. Jest ona dostępna w dokumentacji OpenAI na stronie *https://platform.openai.com/docs/models*. Tutaj skupimy się na opisanych niżej najważniejszych modelach.

InstructGPT

Seria modeli przeznaczonych do wykonywania różnych zadań jednoetapowych. Model `text-ada-001` potrafi realizować jedynie proste operacje, ale jest najszybszym i najtańszym modelem z serii GPT-3. Modele `text-babbage-001` i `text-curie-001` są nieco wydajniejsze, ale też droższe. Model `text-davinci-003` doskonale uzupełnia zdania, ale jest najdroższy z całej serii.

ChatGPT

Model ChatGPT to `gpt-3.5-turbo`. Jest to model konwersacyjny. Danymi wejściowymi jest seria komunikatów, a wynikiem wygenerowany odpowiedni tekst. Model został zaprojektowany pod kątem naprzemiennej konwersacji, ale można go również wykorzystywać do przetwarzania pojedynczych zapytań (nietworzących dialogu). W takich sytuacjach jest pod względem wydajności podobny do modelu `text-davinci-003`, ale dziesięciokrotnie od niego tańszy. Dlatego zaleca się korzystać z niego

w zadaniach jednoetapowych. Kontekst może się składać z maksymalnie 4000 tokenów — taka ich liczba może stanowić dane wejściowe. Dostępny jest również model `gpt-3.5-turbo-16k` o podobnych właściwościach, ale o czterokrotnie większym kontekście.

GPT-4

To największy model opracowany przez OpenAI, multimodalny, wytrenowany na największym korpusie tekstów i obrazów. W efekcie posiadł ekspercką wiedzę w wielu dziedzinach. Potrafi wykonywać skomplikowane instrukcje wydawane w naturalnym języku i dokładnie rozwiązywać trudne problemy. Można go używać zarówno do konwersacji, jak i jednoetapowych zadań, które wykonuje z wysoką precyzją. Dostępne są dwa warianty tego modelu: `gpt-4` i `gpt-4-32k`, z kontekstami obejmującymi odpowiednio 8000 i 32 000 tokenów. W tym drugim przypadku jest to ok. 24 000 słów lub 40 stron tekstu.

Modele GPT-3.5 Turbo i GPT-4 są stale aktualizowane. Nazwy `gpt-3.5-turbo`, `gpt-3.5-turbo-16k`, `gpt-4` i `gpt-4-32k` odnoszą się do najnowszych wersji modeli.

Programiści często potrzebują większej stabilności i przejrzystości modeli LLM wykorzystywanych w aplikacjach. Trudno jest używać modeli językowych, które zmieniają się z dnia na dzień i inaczej reagują na ten sam monit wejściowy. Dlatego są tworzone statyczne migawki modeli. W chwili, gdy pisaliśmy te słowa, aktualne migawki miały nazwy: `gpt-3.5-turbo-0613`, `gpt-3.5-turbo-16k-0613`, `gpt-4-0613` i `gpt-4-32k-0613`.

Jak wspomnieliśmy w rozdziale 1., laboratorium OpenAI zaleca korzystanie z serii modeli InstructGPT, a nie pierwotnej GPT-3. Modele te, oznaczone nazwami `davinci`, `curie`, `babbage` i `ada`, są wciąż dostępne za pomocą interfejsów API. Należy ich jednak używać ostrożnie, ponieważ — jak już wiesz — mogą dawać dziwne, fałszywe lub wprowadzające w błąd odpowiedzi. Są wciąż dostępne, ponieważ są jedynymi, które można dostrajać do danych. Gdy pisaliśmy te słowa, laboratorium OpenAI ogłosiło, że w 2024 r. będzie można dostrajać modele GPT-3.5 Turbo i GPT-4.

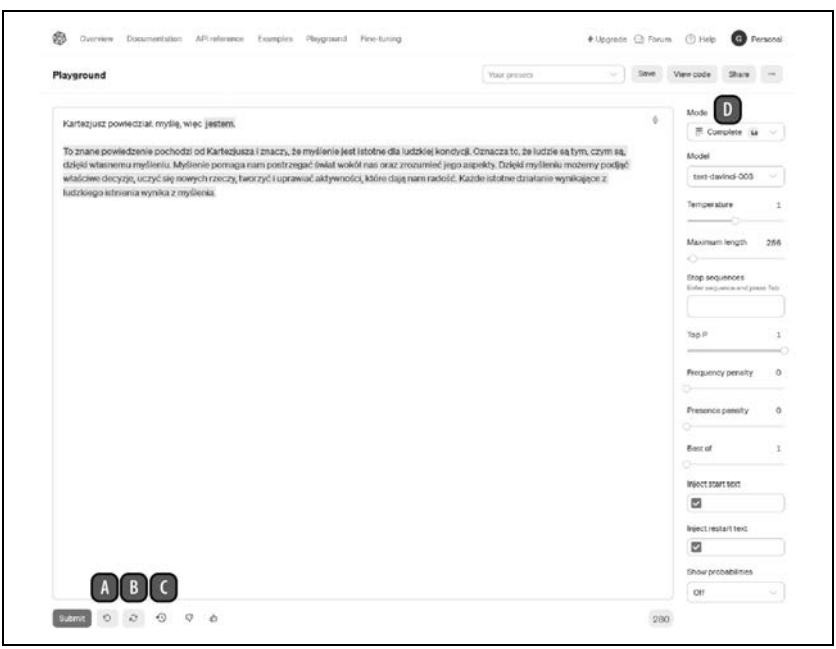
Pod nazwą `davinci-instruct-beta` jest dostępny za pomocą interfejsu API opisany w rozdziale 1. model SFT, uzyskany w wyniku nadzorowanego dostrajania, który nie przeszedł etapu RLHF.

Testowanie modeli GPT za pomocą platformy OpenAI Playground

Doskonałym, bezpośrednim, niewymagającym kodowania sposobem testowania różnych modeli językowych udostępnionych przez laboratorium OpenAI jest użycie platformy Playground. Dzięki niej można szybko przebadać modele LLM pod kątem określonych zastosowań. Wystarczy wpisać monit, wybrać model i sprawdzić wygenerowaną odpowiedź.

Aby dostać się do platformy Playground, wykonaj następujące kroki:

1. Otwórz główną stroną OpenAI (*https://openai.com*) i kliknij odnośnik *Log in* (zaloguj) w prawym górnym rogu.
2. Jeżeli masz już konto, zaloguj się. W przeciwnym wypadku utwórz je, abyś oprócz platformy Playground mógł korzystać z innych dostępnych funkcji. W tym celu kliknij odnośnik *Sign up* (zarejestruj się).
3. Po zalogowaniu kliknij sekcję *API*, a następnie w menu odnośnik *Playground*. Pojawi się widok jak na rysunku 2.2.



Rysunek 2.2. Platforma OpenAI Playground w trybie uzupełniania tekstu

Opcja *ChatGPT Plus* jest niezależna od interfejsów API. Jeżeli ją subskrybujesz, będziesz dodatkowo obciążany kosztami używania interfejsów.

W głównym polu na środku strony wpisz komunikat wejściowy. Aby go uzupełnić, kliknij przycisk *Submit* (wyślij). Komunikat przedstawiony na rysunku 2.2 brzmi: „Kartezjusz powiedział: myślę, więc". Model uzupełnił go słowem „jestem".

Wokół głównego pola znajdują się liczne opcje. Zacznijmy od samego dołu. Po prawej stronie przycisku *Submit* jest przycisk *Undo last* (cofnij ostatni), oznaczony symbolem *A*. Przycisk ten usuwa ostatni wygenerowany tekst. W tym przykładzie jest to słowo „jestem". Następny przycisk to *Regenerate* (generuj ponownie), oznaczony symbolem *B*, który z powrotem uzupełnia usunięty tekst. Dalej jest przycisk *Show history* (pokaż historię), oznaczony symbolem *C*, wyświetlający historię zapytań z ostatnich 30 dni. Zwróć uwagę, że historię można łatwo wyczyścić, jeżeli jest to konieczne na przykład z powodu prywatności danych.

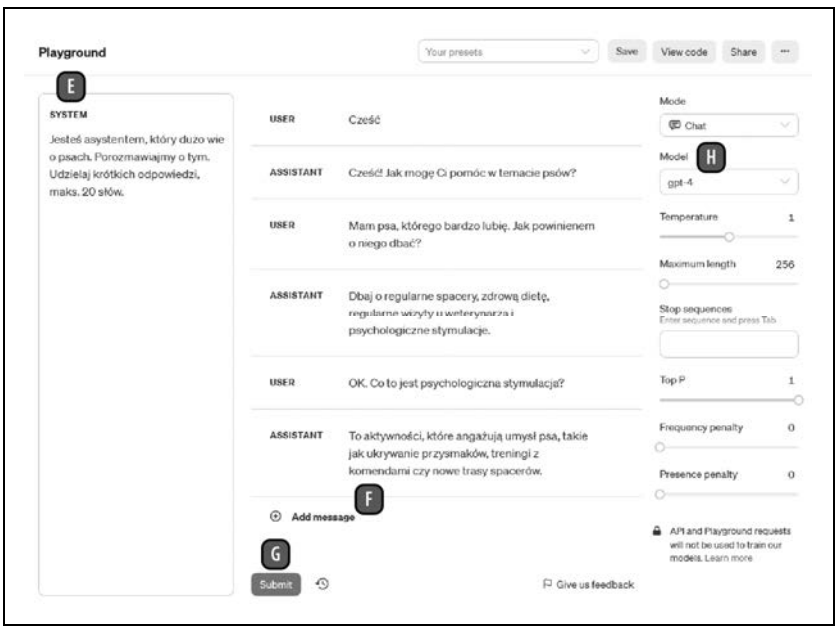
Po prawej stronie ekranu znajduje się panel z ustawieniami interfejsu i wybranego modelu. Poniżej są opisane tylko niektóre z nich. Pozostałe będą omówione w dalszej części książki. Na samej górze znajduje się rozwijana lista *Mode* (tryb), oznaczona symbolem *D*. W chwili, gdy pisaliśmy ten tekst, były dostępne tryby *Chat* (konwersacja, domyślny), *Complete* (uzupełnianie) i *Edit* (edycja).

Tryby *Complete* i *Edit* są oznaczone jako *Legacy* (przestarzałe) i prawdopodobnie w styczniu 2024 r. zostaną usunięte.

Jak widać, model językowy w trybie *Complete* stara się płynnie uzupełniać wprowadzane monity.

Rysunek 2.3 przedstawia platformę Playground w trybie *Chat*. Po lewej stronie znajduje się panel *System*, oznaczony symbolem *E*. Tutaj możesz opisać, jak model ma się zachowywać podczas konwersacji. W tym przykładzie ma ona dotyczyć psów, a ponadto odpowiedzi mają być krótkie. Na środku strony widoczny jest dialog spełniający zadane warunki.

Konwersację z modelem prowadzi się, klikając odnośnik *Add message* (dodaj komunikat, oznaczenie *F*), wpisując pytania i klikając przycisk *Submit* (*G*). Po prawej stronie (*H*) wybiera się model. W tym przypadku jest to GPT-4. Zwróć uwagę, że nie wszystkie modele są dostępne w poszczególnych trybach. Na przykład w trybie *Chat* można wybierać tylko modele z serii GPT-4 i GPT-3.5 Turbo.



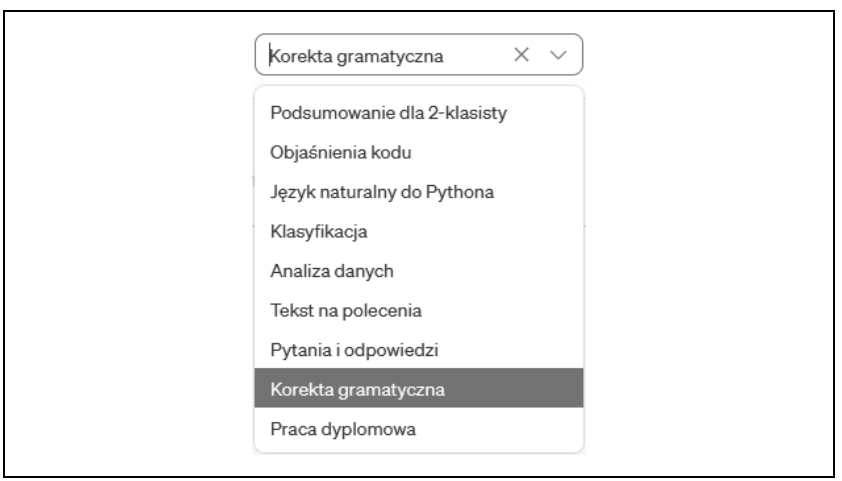
Rysunek 2.3. Platforma OpenAI Playground w trybie konwersacji

Trzecim trybem platformy Playground jest *Edit* (edycja), pokazany na rysunku 2.4. W tym trybie wpisuje się tekst (*I*) i instrukcje (*J*). Model próbuje zmienić tekst zgodnie z instrukcją. W tym przykładzie jest to zdanie o młodym człowieku, który wybiera się w podróż. Model dostał instrukcję, aby zmienić postać na starszą panią. Jak widać, wynik jest zgodny z oczekiwaniami (*K*).

Po prawej stronie, pod rozwijaną listą *Mode*, znajduje się lista *Model* (*L*). Jak już wiesz, tutaj wskazuje się model LLM. Wybór modeli zależy od trybu. Poniżej są widoczne inne parametry określające zachowanie modelu, na przykład *Temperature* (temperatura, *M*). Na razie nie będziemy się zagłębiać w ich szczegóły. Większość z nich poznasz, gdy dokładniej przyjrzymy się funkcjonowaniu poszczególnych modeli.

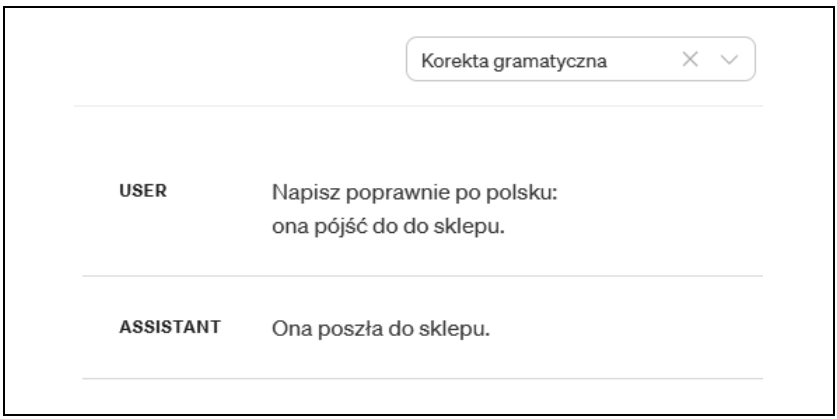
Rysunek 2.4. Platforma OpenAI Playground w trybie edycji

W górnej części strony jest widoczna rozwijana lista *Your presets* (Twoje ustawienia wstępne, *N*) i cztery przyciski. Na rysunku 2.2 model LLM został użyty do uzupełnienia zdania „Kartezjusz powiedział: myślę, więc", ale można mu zlecić wykonywanie innych operacji na wprowadzanych monitach. Rysunek 2.5 przedstawia przykładową listę wstępnych ustawień modelu pod kątem różnych zadań.



Rysunek 2.5. Rozwijana lista wstępnych ustawień modelu

Zwróć uwagę, że wstępne ustawienia obejmują nie tylko instrukcje dla modelu, ale również parametry widoczne po prawej stronie. Na przykład po wybraniu *Korekta gramatyczna* w głównej części ekranu pojawi się monit jak na rysunku 2.6.



Rysunek 2.6. Monit uzyskany z przykładowymi ustawieniami wstępnymi „Korekta gramatyczna”

Po kliknięciu przycisku *Submit* pojawiła się odpowiedź „Ona poszła do sklepu”. Ustawienia zawarte w rozwijanej liście można traktować jako punkty wyjścia. Wszystkie można modyfikować i dostosowywać do własnych potrzeb. Na stronie *https://platform.openai.com/examples* jest dostępna lista przykładowych zastosowań.

Po prawej stronie listy ustawień wstępnych znajduje się przycisk *Save* (zapisz, *O*). Załóżmy, że przygotowałeś monit odpowiedni do swojego zadania oraz parametry modelu i chcesz je później szybko stosować w platformie Playground. Przycisk *Save* zapisuje bieżące parametry jako ustawienia wstępne. Można je opatrzyć opisem. Po zapisaniu pojawią się w rozwijanej liście.

Przedostatni przycisk w górnej części strony to *View code* (zobacz kod, *P*). Po jego kliknięciu pojawi się kod wykonanego testu, który można bezpośrednio wykorzystać w skrypcie. Do wyboru są formaty Python, Node.js, JSON i curl. Ten ostatni umożliwia bezpośrednią interakcję z serwerem OpenAI za pomocą terminala Linux. W przykładzie z uzupełnianiem zdania kod w języku Python ma następującą postać:

```
import os
import openai
openai.api_key = os.getenv("OPENAI_API_KEY")
response = openai.Completion.create(
   model="text-davinci-003",
   prompt="Kartezjusz powiedział: myślę, więc",
   temperature=1,
   max_tokens=256,
   top_p=1,
   frequency_penalty=0,
   presence_penalty=0
)
```

Teraz, gdy już wiesz, jak używać platformy Playground do testowania modeli językowych OpenAI bez kodowania, dowiedz się, jak uzyskać klucze do interfejsów API i zarządzać nimi.

Pierwsze kroki: biblioteka OpenAI dla języka Python

W tym podrozdziale użyjesz klucza API w niewielkim skrypcie w języku Python i wykonasz pierwszy test.

Laboratorium OpenAI udostępnia modele GPT-4 i ChatGPT jako usługi. Oznacza to, że użytkownik nie ma bezpośredniego dostępu do kodów modeli i nie może ich uruchamiać na własnym serwerze. Wdrażaniem modeli i ich zarządzaniem zajmuje się laboratorium OpenAI, natomiast użytkownik może się do nich odwoływać, o ile ma konto i poufny klucz.

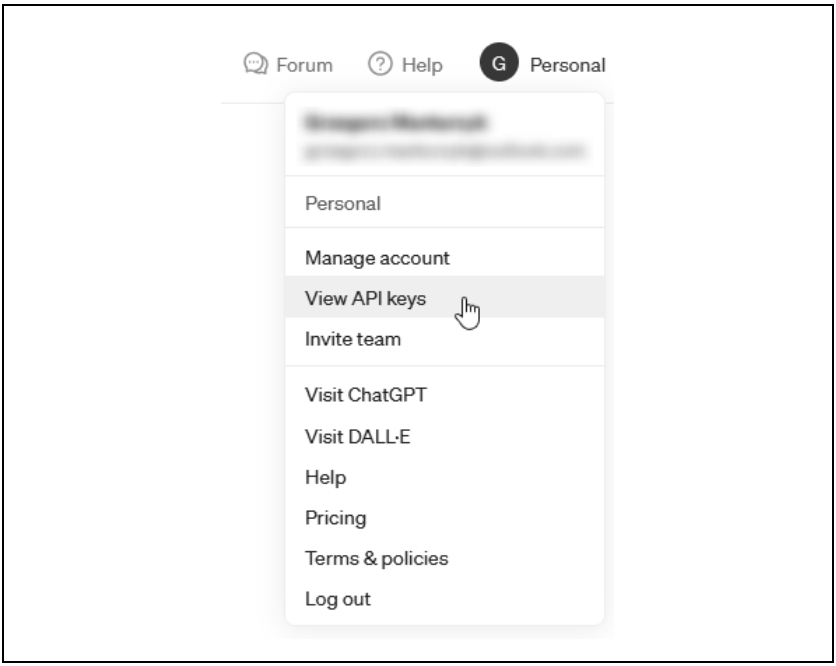
Zanim wykonasz opisane niżej kroki, zaloguj się na stronie OpenAI (*https://platform.openai.com/login?launch*).

Dostęp do modeli i klucz API

Aby móc korzystać z usług OpenAI, należy mieć klucz API. Klucz ten ma dwie funkcje:

- umożliwia wysyłanie zapytań do interfejsów API,
- wiąże zapytania z kontem do celów rozliczeniowych.

Klucz jest niezbędny, aby Twoja aplikacja mogła się odwoływać do usług OpenAI. Aby go uzyskać, zaloguj się na stronie OpenAI (*https://platform.openai.com/login?launch*), kliknij nazwę swojego konta widoczną w prawym górnym rogu strony, a następnie opcję *View API keys* (zobacz klucze API), jak na rysunku 2.7.



Rysunek 2.7. Dostęp do kluczy API

Pojawi się strona *API keys*. Kliknij przycisk *Create new secret key* (utwórz nowy poufny klucz), utwórz klucz i zanotuj go. Klucz jest długim ciągiem znaków rozpoczynającym się od *sk-*.

Przechowuj klucz w bezpiecznym miejscu, ponieważ jest on bezpośrednio powiązany z Twoim kontem. Ujawnienie go może narazić Cię na niepożądane koszty.

Dobrą praktyką jest zapisanie klucza w zmiennej środowiskowej. Dzięki temu aplikacja będzie mieć do niego dostęp bez konieczności zapisywania go bezpośrednio w kodzie. W tym celu wykonaj opisane niżej polecenia.

W systemie Linux lub macOS:

```
# Definicja zmiennej środowiskowej OPENAI_API_KEY w bieżącej sesji:
export OPENAI_API_KEY=sk-...
# Sprawdzenie, czy zmienna została utworzona:
echo $OPENAI_API_KEY
```

W systemie Windows:

```
# Definicja zmiennej środowiskowej OPENAI_API_KEY w bieżącej sesji:
set OPENAI_API_KEY=sk-...
# Sprawdzenie, czy zmienna została utworzona:
echo %OPENAI_API_KEY%
```

Powyższe polecenia definiują zmienną środowiskową i udostępniają klucz procesom uruchamianym w bieżącej sesji powłoki. W systemie Linux kod można wpisać bezpośrednio w pliku *.bashrc*. Zmienna środowiskowa będzie wtedy dostępna we wszystkich sesjach. Oczywiście, nie można tych poleceń umieszczać w kodzie wysyłanym do publicznego repozytorium.

Aby utworzyć trwałą zmienną środowiskową w systemie Windows, naciśnij jednocześnie klawisze *Windows* i *R*. W oknie *Uruchom*, które się pojawi, wpisz *sysdm.cpl*, aby otworzyć okno *Właściwości systemu*. Kliknij zakładkę *Zaawansowane*, a następnie przycisk *Zmienne środowiskowe*. W oknie, które się pojawi, zdefiniuj nową zmienną i przypisz jej klucz API.

Na stronie OpenAI (*https://help.openai.com/en/articles/5112595-↪best-practices-for-api-key-safety*) znajdziesz szczegółowe informacje dotyczące bezpieczeństwa kluczy.

Teraz, gdy masz już klucz, czas napisać pierwszy program „Witaj, świecie!" wykorzystujący interfejs OpenAI API.

Przykład „Witaj, świecie!"

W tym podrozdziale napiszesz kilka pierwszych wierszy kodu wykorzystującego bibliotekę OpenAI dla języka Python. Zaczniesz od klasycznego przykładu „Witaj, świecie!", dzięki któremu dowiesz się, jak funkcjonują usługi OpenAI.

Za pomocą polecenia `pip` zainstaluj bibliotekę OpenAI:

```
pip install openai
```

Następnie napisz kod odwołujący się do interfejsu API:

```
import openai
# Wywołaj punkt końcowy ChatCompletion modelu ChatGPT:
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[{"role": "user", "content": "Witaj, świecie!"}],
```

```
)
# Wyodrębnij odpowiedź:
print(response["choices"][0]["message"]["content"])
```

Uzyskasz następujący wynik:

```
Cześć! Jak mogę ci pomóc?
```

Gratulacje! Napisałeś pierwszy program wykorzystujący bibliotekę OpenAI dla języka Python. Teraz dowiedz się dokładnie, jak z niej korzystać.

Biblioteka zawiera również narzędzie terminalowe. Poniższe polecenie, wpisane w terminalu, daje ten sam wynik co przykładowy kod:

```
openai api chat_completions.create -m gpt-3.5-turbo \
    -g user "Witaj, świecie!"
```

Ponadto do interfejsu OpenAI API można się odwoływać bezpośrednio, wysyłając zapytania HTTP, lub pośrednio, za pomocą oficjalnej biblioteki Node.js lub innych bibliotek rozwijanych przez społeczność użytkowników (*https://platform.openai.com/docs/libraries*).

Jak zauważyłeś, w przykładowym kodzie nie jest jawnie użyty klucz OpenAI API. Nie ma takiej potrzeby, ponieważ biblioteka OpenAI wyszukuje zmienną środowiskową `OPENAI_API_KEY`. Innym rozwiązaniem jest użycie następującego kodu, który wskazuje modułowi `openai` plik z kluczem:

```
# Załaduj klucz API z pliku:
openai.api_key_path = <ścieżka do pliku>
```

Można też jawnie załadować klucz w następujący sposób:

```
# Załaduj klucz API:
openai.api_key = os.getenv("OPENAI_API_KEY")
```

Zalecamy stosowanie ogólnie przyjętej konwencji ze zmienną środowiskową. Klucz należy zapisać w pliku *.env*. Plik ten jest wskazany w pliku *.gitignore*, aby nie był wysyłany do systemu kontroli kodu źródłowego. W kodzie należy wywołać funkcję `load_dotenv` ładującą zmienne środowiskowe i zaimportować bibliotekę *openai*, jak niżej:

```
from dotenv import load_dotenv
load_dotenv()
import openai
```

Ważne jest, aby po instrukcji ładującej plik *.env* umieścić wiersz `import openai`. W przeciwnym wypadku ustawienia biblioteki OpenAI nie zostaną poprawnie zastosowane.

Po zapoznaniu się z podstawami modeli ChatGPT i GPT-4 możemy przejść do szczegółów ich użycia.

Korzystanie z modeli ChatGPT i GPT-4

W tym podrozdziale dowiesz się, jak korzystać z modeli ChatGPT i GPT-4 za pośrednictwem biblioteki OpenAI dla języka Python. Gdy pisaliśmy te słowa, najtańszy i najbardziej funkcjonalny był model GPT 3.5 Turbo. Dlatego w większości przypadków jest to najlepszy wybór. Poniższy kod przedstawia przykład jego użycia:

```
import openai
# W modelu GPT 3.5 Turbo punkt końcowy ma nazwę ChatCompletion.
openai.ChatCompletion.create(
    # Model GPT 3.5 Turbo ma nazwę gpt-3.5-turbo.
    model="gpt-3.5-turbo",
    # Konwersacja jest listą komunikatów.
    messages=[
        {"role": "system", "content": "Jesteś dobrym nauczycielem."},
        {
            "role": "user",
            "content": "Czy istnieją inne wskaźniki złożoności algorytmu
            ↪niż czas?",
        },
        {
            "role": "assistant",
            "content": "Tak, oprócz złożoności czasowej algorytmu istnieją \
            inne miary, na przykład złożoność przestrzenna.",
        },
        {"role": "user", "content": "Co to jest?"},
    ],
)
```

Powyższy przykład wykorzystuje minimalną liczbę parametrów. Wskazane zostały jedynie model LLM i komunikaty wejściowe. Jak widać, format danych pozwala wysyłać do modelu wieloetapową konwersację. Należy pamiętać, że interfejs API nie przechowuje w swoim kontekście użytych wcześniej komunikatów. Pytanie „Co to jest?” odnosi się do ostatniej odpowiedzi i ma sens tylko wtedy, gdy model tę odpowiedź zna. Dlatego aby symulować konwersację, należy ją za każdym razem wysyłać w całości. Ten temat dokładniej opiszemy w następnym podrozdziale.

Modele GPT 3.5 Turbo i GPT-4 są zoptymalizowane pod kątem konwersacji, ale nie jest to ich jedyne zastosowanie. Oba można wykorzystywać do zadań jedno- i wieloetapowych. Sprawdzają się w tradycyjnych zadaniach uzupełniania tekstu. Wystarczy w tym celu podać monit z prośbą o uzupełnienie.

Punkty końcowe w obu modelach mają taką samą nazwę: `openai.ChatCompletion`. Zmieniając identyfikator, można się przełączać między modelami GPT-3.5 Turbo i GPT-4. Oprócz tego nie trzeba wprowadzać w kodzie żadnych modyfikacji.

Parametry wejściowe punktu końcowego ChatCompletion

Przyjrzyjmy się dokładniej, jak używać punktu końcowego `openai.ChatCompletion` i jego metody `create()`.

Metoda `create()` służy do odwoływania się do modeli. Oprócz niej istnieją inne metody, które nie służą do interakcji. Kod biblioteki OpenAI dla języka Python jest dostępny w serwisie GitHub (*https://github.com/openai/openai-python/blob/main/src/openai/resources/chat/completions.py*).

Wymagane parametry wejściowe

Metoda `create()` punktu końcowego `openai.ChatCompletion` ma wiele parametrów, z których wymagane są tylko dwa, opisane w tabeli 2.1.

Tabela 2.1. Wymagane parametry wejściowe

Parametr	Typ	Opis
`model`	Ciąg znaków	Identyfikator wykorzystywanego modelu. W chwili, gdy pisaliśmy ten tekst, były dostępne modele o identyfikatorach: `gpt-3.5-turbo`, `gpt-3.5-turbo-0301`, `gpt-3.5-turbo-0613`, `gpt-3.5-turbo-16k`, `gpt-3.5-turbo-16k-0613`, `gpt-3.5-turbo-instruct`, `gpt-3.5-turbo-instruct-0914`, `gpt-4`, `gpt-4-0314` i `gpt-4-0613`. Listę dostępnych modeli można uzyskać, wywołując inny punkt końcowy i metodę z biblioteki OpenAI: `openai.Model.list()`. Pamiętaj, że nie każdy model posiada punkt końcowy `openai.ChatCompletion`.
`messages`	Tablica	Tablica obiektów komunikatów tworzących konwersację. Obiekt komunikatu ma dwa atrybuty: `role`, o dopuszczalnych wartościach `system`, `user` lub `assistant`, oraz `content`, którego wartością jest treść komunikatu.

Konwersację rozpoczyna opcjonalny komunikat systemowy. Po nim następują naprzemiennie komunikaty użytkownika i asystenta. Komunikat systemowy określa, jak asystent ma się zachowywać. Komunikaty użytkownika są odpowiednikami pytań lub stwierdzeń wpisywanych przez człowieka w interfejsie przeglądarkowym modelu ChatGPT. Mogą być definiowane przez użytkownika aplikacji lub zapisane w kodzie i pełnić rolę instrukcji dla modelu.

Komunikaty asystenta mają dwie funkcje: przechowują odpowiedzi niezbędne do kontynuowania konwersacji lub opisują pożądane zachowania modelu. Modele nie mają pamięci, w której mogłyby przechowywać wcześniejsze komunikaty. Dlatego trzeba je podawać, aby dostarczyć modelowi kontekst konwersacji i wszystkie istotne informacje.

Długość konwersacji i liczba tokenów

Jak już wiesz, długość konwersacji przekłada się na liczbę tokenów, która ma wpływ na:

- **koszt** — płaci się za tokeny,
- **czas** — im więcej tokenów, tym dłuższy czas odpowiedzi, który może sięgać kilku minut,
- **działanie modelu** — całkowita liczba tokenów nie może przekraczać limitu właściwego dla danego modelu. Przykładowe limity tokenów znajdują się w podrozdziale „Uwagi”.

Jak widać, należy uważnie kontrolować długość konwersacji. Liczbę tokenów wejściowych ogranicza się, skracając komunikaty, a tokenów wyjściowych, nadając odpowiednią wartość parametrowi `max_tokens`, opisanemu w następnym podrozdziale.

Laboratorium OpenAI udostępnia bibliotekę *tiktoken* (*https://github.com/openai/tiktoken*), za pomocą której można sprawdzić, z ilu tokenów składa się zadany ciąg znaków. Zdecydowanie zalecamy, aby korzystać z tej biblioteki i szacować koszty przed wywołaniem punktu końcowego.

Opcjonalne parametry wejściowe

Oprócz parametrów obowiązkowych istnieje wiele opcjonalnych, umożliwiających dostosowanie interakcji z modelem. Nie będziemy ich wszystkich tutaj opisywać. Tabela 2.2 zawiera kilka najważniejszych.

Tabela 2.2. Wybrane dodatkowe parametry opcjonalne

Parametr	Typ	Opis
`functions`	Tablica	Tablica dostępnych funkcji. Informacje na temat stosowania funkcji znajdują się w podrozdziale „Od uzupełniania tekstu do funkcji”.
`function_call`	Ciąg znaków lub obiekt	Parametr określający wymagane odpowiedzi modelu: Wartość `none` powoduje, że model odpowiada w zwykły sposób. Wartość `{"name":"nazwa_funkcji"}` powoduje, że model odpowiada, wykorzystując zadaną funkcję. Wartość `auto` powoduje, że model odpowiada w zwykły sposób lub wykorzystuje funkcje określone w parametrze `functions`.
`temperature`	Liczba rzeczywista (od 0 do 2, domyślnie 1)	Wartość 0 powoduje, że za każdym razem uzupełnienie danych wejściowych jest takie samo. Model generuje spójne odpowiedzi, ale nie musi działać deterministycznie. Im większa wartość tego parametru, tym bardziej rozbieżne są odpowiedzi. Model generuje odpowiedź, prognozując kolejne tokeny. Na podstawie kontekstu wejściowego przypisuje tokenom wartości prawdopodobieństwa. Parametr o wartości 0 powoduje, że model zawsze wybiera token o największym prawdopodobieństwie. Większe wartości parametru pozwalają uzyskiwać bardziej urozmaicone i kreatywne odpowiedzi.
`n`	Liczba całkowita (domyślnie 1)	Parametr umożliwiający generowanie kilku uzupełnień zadanego komunikatu wejściowego. Jeżeli jednak parametr `temperature` ma wartość 0, wówczas wszystkie uzupełnienia są identyczne lub bardzo do siebie podobne.
`stream`	Wartość logiczna (domyślnie `false`)	Jak sugeruje nazwa, parametr powoduje generowanie odpowiedzi w formie strumienia, tj. stopniowe wysyłanie kolejnych części komunikatu, jak w interfejsie przeglądarkowym modelu ChatGPT. W przypadku dłuższych uzupełnień zapewnia to użytkownikowi lepsze wrażenia.
`max_tokens`	Liczba całkowita	Parametr określający maksymalną liczbę tokenów składających się na uzupełnienie komunikatu. Jest opcjonalny, ale dobrą praktyką jest stosowanie go, ponieważ pozwala kontrolować koszty. Pamiętaj, że może nie być uwzględniany, jeżeli jego wartość będzie bardzo duża. Maksymalną łączną liczbę tokenów wejściowych i wyjściowych określa limit danego modelu.

Więcej parametrów i szczegółowych informacji znajdziesz w oficjalnej dokumentacji (*https://platform.openai.com/docs/api-reference/chat*).

Format odpowiedzi punktu końcowego ChatCompletion

Teraz, gdy wiesz, jakie informacje są potrzebne do wysłania zapytania do modelu konwersacyjnego, dowiedz się, co zawiera wynik. Poniższy kod przedstawia odpowiedź z przykładu „Witaj, świecie!”:

```
{
  "id": "chatcmpl-8GeuXOQi1aP4OfvEiv3RGEIBc2y57",
  "object": "chat.completion",
  "created": 1699006637,
  "model": "gpt-3.5-turbo-0613",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Cze\u015b\u0107! Jak mog\u0119 ci pom\u00f3c?"
      },
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 15,
    "completion_tokens": 12,
    "total_tokens": 27
  }
}
```

Tabela 2.3 zawiera szczegółowy opis wyniku.

Jeżeli użyjesz parametru `n` o wartości większej niż 1, atrybut `choices` będzie zawierał kilka odpowiedzi. Atrybut `prompt_tokens` nie ulegnie zmianie, a `complete_tokens` będzie miał wartość mniej więcej `n`-krotnie większą.

Od uzupełniania tekstu do funkcji

Modele OpenAI mogą zwracać odpowiedzi w formacie JSON, zawierające argumenty dla wywoływanych później funkcji. Model nie wywołuje funkcji samodzielnie, może jednak konwertować odpowiedzi na format umożliwiający

Tabela 2.3. Opis wyniku wygenerowanego przez model uzupełniający komunikaty

Atrybut	Typ	Opis
`id`	Ciąg znaków	Identyfikator wykorzystywany wewnętrznie przez model.
`object`	Ciąg znaków	W przypadku modeli GPT-4 i GPT-3.5 atrybut ten ma zawsze wartość `chat.completion` oznaczającą punkt końcowy `ChatCompletion`.
`created`	Znacznik czasu	Znacznik czasu wygenerowania odpowiedzi. W przykładzie „Witaj, świecie!" jest to piątek, 3 listopada 2023 r., godz. 11:17:17.
`model`	Ciąg znaków	Identyfikator użytego modelu, taki sam jak użyty w danych wejściowych.
`choices`	Tablica obiektów	Tablica zawierająca odpowiedź modelu. Domyślnie zawiera tylko jeden element, ale tę liczbę można zmienić za pomocą parametru `n` (podrozdział „Opcjonalne parametry wejściowe"). Element tablicy zawiera następujące atrybuty: `finish_reason`: ciąg znaków zawierający powód udzielenia odpowiedzi. W przykładzie „Witaj, świecie!" jest to ciąg `stop` oznaczający, że model wygenerował pełną odpowiedź. W przypadku problemu atrybut ten zawiera odpowiedni komunikat. `index`: liczba całkowita oznaczająca indeks obiektu w tablicy `choices`. `message`: obiekt zawierający atrybuty `role` oraz `content` lub `function_call`. Atrybut `role` zawsze zawiera ciąg `assistant`, a `content` tekst wygenerowany przez model, który zazwyczaj wyodrębnia się za pomocą wyrażenia `response['choices'][0]['message']['content']`. Szczegółowe informacje zawiera podrozdział „Od uzupełniania tekstu do funkcji".
`usage`	Ciąg znaków	Informacje o liczbie tokenów wykorzystanych w zapytaniu, a więc o kosztach. Atrybut `prompt_tokens` zawiera liczbę tokenów w komunikacie wejściowym, `completion_tokens` w wyjściowym, a `total_tokens` — jak się można domyślić — sumę `prompt_tokens` i `completion_tokens`.

przetwarzanie ich przez kod użytkownika. Zamiast budować skomplikowany monit określający format umożliwiający przetwarzanie odpowiedzi za pomocą kodu, można zdefiniować funkcję, która będzie konwertować język naturalny na odwołania do interfejsu API lub zapytania do bazy danych, wyodrębniać z tekstu strukturalne dane, wywoływać zewnętrzne narzędzia i generować odpowiedzi.

Zgodnie z tabelą 2.2, zawierającą parametry wejściowe dla punktu końcowego uzupełniającego komunikaty, definicje funkcji umieszcza się w parametrze `functions` w formie tablicy obiektów funkcji. Tabela 2.4 zawiera opis atrybutów takiego obiektu.

Tabela 2.4. Atrybuty obiektu funkcji

Atrybut	Typ	Opis
`name`	Ciąg znaków (wymagany)	Nazwa funkcji.
`description`	Ciąg znaków	Opis funkcji.
`parameters`	Obiekt	Parametry funkcji w formacie JSON (*http://json-schema.org*).

Załóżmy, że mamy bazę danych produktów firmy. Zdefiniujmy funkcję, która wyszukuje w niej określone informacje:

```
# Przykładowa funkcja
def find_product(sql_query):
    # Wysłanie zapytania do bazy
    results = [
        {"name": "pióro", "color": "niebieskie", "price": 199},
        {"name": "pióro", "color": "czerwone", "price": 178},
    ]
    return results
```

W parametrze `functions` umieszczamy następującą specyfikację funkcji:

```
# Definicja funkcji
functions = [
    {
        "name": "find_product",
        "description": "Utworzenie listy produktów na podstawie zapytania
        ↪SQL",
        "parameters": {
            "type": "object",
            "properties": {
                "sql_query": {
                    "type": "string",
                    "description": "Zapytanie SQL",
                }
            },
            "required": ["sql_query"],
        },
    }
]
```

Następnie tworzymy konwersację i odwołujemy się do punktu końcowego `openai.ChatCompletion`:

```
# Przykładowe pytanie
user_question = "Potrzebuję dwóch najlepszych produktów w cenie do 200 zł"
messages = [{"role": "user", "content": user_question}]
# Odwołanie do punktu końcowego openai.ChatCompletion z użyciem definicji funkcji
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo-0613", messages=messages, functions=functions
)
response_message = response["choices"][0]["message"]
messages.append(response_message)
```

Model tworzy zapytanie SQL do wykorzystania. Wartość atrybutu `function_call` odpowiedzi jest następująca:

```
"function_call": {
    "name": "find_product",
    "arguments": '{\n  "sql_query": "SELECT * FROM products \
        WHERE price < 2.00 ORDER BY price ASC LIMIT 2"\n}',
    }
```

Następnie wywołujemy funkcję i kontynuujemy konwersację, wykorzystując uzyskany wynik:

```
# Wywołanie funkcji
function_args = json.loads(
    response_message["function_call"]["arguments"]
)
products = find_product(function_args.get("sql_query"))
# Dołączenie wyniku funkcji do komunikatu
messages.append(
    {
        "role": "function",
        "name": function_name,
        "content": json.dumps(products),
    }
)
# Przekształcenie wyniku funkcji na język naturalny
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo-0613",
    messages=messages,
)
```

Na koniec wyodrębniamy ostateczną odpowiedź i uzyskujemy następujący wynik:

```
Oto dwa najlepsze produkty w cenie do 200 zł:
1. Pióro niebieskie - cena: 199 zł
2. Pióro czerwone - cena: 178 zł
Oba te produkty mają wysokie oceny i są dostępne w cenie do 200 zł.
```

Ten prosty przykład pokazuje, jak za pomocą funkcji można tworzyć rozwiązania umożliwiające użytkownikom komunikowanie się z bazą danych w języku naturalnym. Dzięki definicjom funkcji można sterować modelem tak, aby odpowiadał dokładnie według wymagań, jak również można integrować odpowiedzi z aplikacją.

Korzystanie z innych modeli uzupełniających tekst

Jak wspomnieliśmy wcześniej, laboratorium OpenAI oferuje oprócz GPT-3 i GPT-3.5 kilka innych modeli, wykorzystujących inne punkty końcowe niż modele ChatGPT i GPT-4. Model GPT 3.5 Turbo jest zazwyczaj najlepszym wyborem zarówno pod względem ceny, jak i wydajności. Warto jednak wiedzieć, jak korzystać z innych modeli uzupełniających komunikaty, szczególnie w takich zastosowaniach jak dostrajanie, gdzie model GPT-3 jest jedyną opcją.

Laboratorium OpenAI opublikowało plan wycofania punktu końcowego do uzupełniania tekstu. Opisaliśmy go tutaj tylko dlatego, że bazowe modele uzupełniające są jedynymi, które można dostrajać. Laboratorium udostępni rozwiązanie do dostrajania modeli konwersacyjnych do stycznia 2024 r. W chwili, gdy pisaliśmy ten tekst, na temat tego narzędzia nie było dostępnych żadnych informacji, więc nie mogliśmy go opisać.

Pomiędzy modelami uzupełniającymi tekst i konwersację jest istotna różnica. Jak wiadomo, oba generują teksty, ale model uzupełniający konwersację jest zoptymalizowany pod kątem tego zadania. Jak widać w poniższym kodzie, główna różnica w porównaniu z punktem końcowym `openai.ChatCompletion` leży w formacie danych wejściowych. W modelu uzupełniającym komunikat monit nie zawiera konwersacji, tylko pojedynczy tekst.

```
import openai
# Odwołanie do punktu końcowego openai.Completion
response = openai.Completion.create(
    model="text-davinci-003", prompt="Witaj, świecie!"
)
# Wyodrębnienie odpowiedzi
print(response["choices"][0]["text"])
```

Powyższy kod wyświetla uzupełnienie podobne do poniższego:

```
Cześć! Miło Cię poz
```

Następny punkt zawiera szczegółowe informacje o parametrach wejściowych punktu końcowego uzupełniającego tekst.

Parametry wejściowe punktu końcowego Completion

Parametry wejściowe metody `create()` punktu końcowego `openai.Completion` są bardzo podobne jak w opisanym wcześniej punkcie `openai.ChatCompletion`. W tym punkcie przedstawiamy najważniejsze parametry oraz kwestie dotyczące długości monitu.

Najważniejsze parametry wejściowe

Tabela 2.5 zawiera wymagane parametry wejściowe i kilka opcjonalnych, naszym zdaniem najbardziej przydatnych.

Tabela 2.5. Parametry wymagane i opcjonalne punktu końcowego Completion

Parametr	Typ	Opis
`model`	Ciąg znaków	Identyfikator wykorzystywanego modelu (tak jak w punkcie końcowym `openai.ChatCompletion`). Jest to jedyny wymagany parametr.
`prompt`	Ciąg znaków lub tablica (domyślnie `<|endoftext|>`)	Uzupełniany monit. Ten parametr stanowi główną różnicę pomiędzy punktami końcowymi `openai.ChatCompletion` a `openai.Completion`. Jego wartością może być ciąg znaków, tablica ciągów znaków, tablica tokenów lub tablica tablic tokenów. Jeżeli nie zostanie określony, model wygeneruje tekst rozpoczynający nowy dokument.
`max_tokens`	Liczba całkowita (domyślnie 16)	Maksymalna liczba tokenów, z których może się składać uzupełnienie tekstu. Domyślna wartość może być za mała w pewnych sytuacjach. Dlatego należy ją odpowiednio zmienić.
`suffix`	Ciąg znaków (domyślnie `null`)	Tekst dołączany do uzupełnienia (sufiks). Umożliwia również wstawianie tekstu.

Długość monitu i liczba tokenów

Podobnie jak w przypadku modelu konwersacyjnego, opłaty za korzystanie z modelu uzupełniającego zależą od ilości danych wejściowych i wyjściowych. Dlatego należy zwracać uwagę na długości parametrów `prompt` i `suffix` (jeżeli jest używany). Aby uniknąć przykrych niespodzianek, wielkości uzyskiwanych wyników należy kontrolować za pomocą parametru `max_tokens`.

Opcjonalne parametry wejściowe

Podobnie jak w przypadku punktu modelu konwersacyjnego, funkcjonowanie modelu uzupełniającego można dostrajać za pomocą parametrów opcjonalnych. Są takie same jak w punkcie końcowym `openai.ChatCompletion`, więc nie będziemy tutaj ich ponownie opisywać. Pamiętaj, że wyniki można kontrolować za pomocą parametrów `temperature` lub `n`, a koszty przy użyciu parametru `max_tokens`. Parametr `stream` zapewnia użytkownikowi lepsze wrażenia w przypadku długich uzupełnień.

Format odpowiedzi punktu końcowego Completion

Teraz, gdy posiadasz wszystkie informacje potrzebne do wysyłania zapytań do modelu uzupełniającego, przekonaj się, że wyniki są bardzo podobne do generowanych przez model konwersacyjny. Poniższy listing przedstawia wynik z przykładu „Witaj, świecie!” z użyciem modelu `text-davinci-003`:

```
{
  "id": "cmpl-8HDgB1dCPdDHIntfyesjxJyT85Eaw",
  "object": "text_completion",
  "created": 1699114851,
  "model": "text-davinci-003",
  "choices": [
    {
      "text": "\n\nCze\u015b\u0107! Mi\u0142o Ci\u0119 poz",
      "index": 0,
      "logprobs": null,
      "finish_reason": "length"
    }
  ],
  "usage": {
    "prompt_tokens": 11,
    "completion_tokens": 16,
    "total_tokens": 27
  }
}
```

Powyższy wynik jest bardzo podobny do wygenerowanego przez model konwersacyjny. Jedyna różnica leży w atrybucie `choices`, który nie ma atrybutów `message`, `content` ani `role`, tylko zwykły atrybut `text` zawierający wygenerowane uzupełnienie.

Uwagi

Zanim zaczniesz intensywnie korzystać z interfejsów API, rozważ dwie ważne kwestie: koszty i prywatność danych.

Ceny i limity tokenów

Cennik usług OpenAI jest dostępny na stronie *https://openai.com/pricing*. Pamiętaj, że laboratorium OpenAI nie jest zobowiązane do utrzymywania stałych cen i może je w dowolnym momencie zmieniać. Tabela 2.6 przedstawia ceny użytkowania najpopularniejszych modeli w czasie, gdy pisaliśmy te słowa.

Tabela 2.6. Ceny i limity tokenów dla wybranych modeli

Seria modeli	Model	Ceny	Limit tokenów
Konwersacyjne	`gpt-4`	Monit: 0,03 dol. za 1000 tokenów Uzupełnienie: 0,06 dol. za 1000 tokenów	8192
Konwersacyjne	`gpt-4-32k`	Monit: 0,06 dol. za 1000 tokenów Uzupełnienie: 0,12 dol. za 1000 tokenów	32 768
Konwersacyjne	`gpt-3.5-turbo`	Monit: 0,0015 dol. za 1000 tokenów Uzupełnienie: 0,002 dol. za 1000 tokenów	4096
Konwersacyjne	`gpt-3.5-turbo-16k`	Monit: 0,003 dol. za 1000 tokenów Uzupełnienie: 0,004 dol. za 1000 tokenów	16 384
Uzupełniające	`text-davinci-003`	0,02 dol. za 1000 tokenów	4096

Zwróć uwagę na kilka rzeczy w tabeli 2.6. Model `text-davinci-003` jest ponad 10-krotnie droższy od `gpt-3.5-turbo`. Oba modele mogą wykonywać zadania jednoetapowe i są pod tym względem niemal równie dokładne. Dlatego w takich przypadkach zaleca się korzystać z modelu `gpt-3.5-turbo`.

Model GPT-3.5 Turbo jest tańszy niż GPT-4, a w wielu podstawowych zadaniach różnice pomiędzy nimi są nieistotne. Jednak w skomplikowanych zadaniach wnioskowania model GPT-4 sprawdza się znacznie lepiej niż GPT-3.5 Turbo.

Cennik modeli konwersacyjnych jest inny niż modelu uzupełniającego i uwzględnia dane wejściowe (monit) i wyjściowe (uzupełnienie).

Kontekst modelu GPT-4 jest dwukrotnie dłuższy niż GPT-3.5 Turbo i obejmuje ponad 32 000 tokenów, czyli 25 000 słów. Ten pierwszy potrafi realizować takie zadania jak generowanie długich treści, zaawansowane konwersacje oraz wyszukiwanie i analizowanie dokumentów.

Bezpieczeństwo i prywatność danych

W chwili, gdy pisaliśmy te słowa, laboratorium OpenAI twierdziło, że nie wykorzystuje danych wejściowych do ponownego trenowania modeli, chyba że użytkownik wyrazi na to zgodę. Niemniej dane te są przechowywane przez 30 dni na potrzeby sprawdzania, czy modele są wykorzystywane zgodnie z przeznaczeniem. Oznacza to, że pracownicy laboratorium OpenAI i jego kontrahenci mają dostęp do danych wysyłanych przez użytkowników do interfejsów API.

Nigdy nie wysyłaj do punktów końcowych poufnych informacji, takich jak dane osobowe lub hasła. Zalecamy regularne sprawdzanie zasad przetwarzania danych przez OpenAI (*https://openai.com/policies/api-data-usage-policies*), ponieważ mogą się zmieniać. Ponadto wysyłane dane, w tym osobowe, są zapisywane na serwerach w Stanach Zjednoczonych, co może mieć konsekwencje prawne dla użytkowników Twoich aplikacji w innych krajach.

Więcej szczegółowych informacji na temat bezpieczeństwa i prywatności danych w modelach LLM znajdziesz w rozdziale 3.

Inne interfejsy API i ich funkcjonalności

Konto OpenAI daje dostęp do wielu innych funkcjonalności oprócz uzupełniania tekstu. Kilka z nich opisujemy w tym podrozdziale, a ich pełną listę znajdziesz w dokumentacji (*https://platform.openai.com/docs/api-reference*).

Osadzenia

Model sztucznej inteligencji opiera swoje działanie na funkcjach matematycznych, więc wymaga danych liczbowych. Jednak wiele rodzajów danych, na przykład słowa i tokeny, nie jest liczbami. Aby móc je skutecznie przetwarzać, przekształca się je w **osadzenia**, czyli wektory liczb. Osadzenia są potrzebne w pewnych sytuacjach, dlatego laboratorium OpenAI udostępnia model przekształcający teksty w wektory. Za pomocą odpowiedniego punktu końcowego można tworzyć wektorowe reprezentacje wprowadzanych tekstów, które mogą być danymi wejściowymi dla innych modeli i algorytmów przetwarzania języka naturalnego.

W chwili, gdy pisaliśmy te słowa, laboratorium OpenAI zalecało stosować do niemal wszystkich zadań najnowszy model `text-embedding-ada-002`. Jego użycie jest bardzo proste:

```
result = openai.Embedding.create(
    model="text-embedding-ada-002", input="tekst wejściowy"
)
```

Osadzenie odczytuje się w następujący sposób:

```
result['data']['embedding']
```

Wynikowe osadzenie jest tablicą liczb zmiennoprzecinkowych.

Pełny opis osadzeń jest dostępny w dokumentacji (*https://platform.openai.com/docs/api-reference/embeddings*).

Generowanie osadzeń polega na umieszczaniu tekstu w określonej przestrzeni uwzględniającej podobieństwo semantyczne. Osadzenia wykorzystuje się do realizacji następujących zadań:

- **wyszukiwanie**: sortowanie wyników wyszukiwania według ich istotności,
- **rekomendowanie**: proponowanie tekstów powiązanych z zadanym ciągiem znaków,
- **klastrowanie**: grupowanie ciągów znaków według podobieństwa,
- **wykrywanie anomalii**: wyszukiwanie ciągów niepowiązanych z innymi ciągami.

Tłumaczenie języka naturalnego za pomocą osadzeń w uczeniu maszynowym

Osadzenia są wykorzystywane w uczeniu maszynowym, szczególnie w modelach językowych. Reprezentują dane kategorialne, takie jak tokeny (zazwyczaj pojedyncze słowa) lub ich grupy (zdania) w formacie numerycznym, w szczególności w postaci wektorów liczb rzeczywistych. To przekształcenie jest niezbędne, ponieważ modele uczenia maszynowego nie są przystosowane do przetwarzania danych kategorialnych i wymagają danych liczbowych. Generator osadzeń można traktować jako szczególnego rodzaju interpreter języka naturalnego, który tłumaczy bogaty świat słów i zdań na uniwersalny język liczb zrozumiały dla modeli uczenia maszynowego.

Osadzenia mają niezwykłą cechę odzwierciedlania **podobieństw semantycznych**. Oznacza to, że słowa lub frazy o podobnym znaczeniu znajdują się blisko siebie w przestrzeni liczbowej. Ma to fundamentalne znaczenie w procesie **pozyskiwania informacji**, polegającym na wyodrębnianiu istotnych informacji z dużego zbioru danych. Osadzenia, które z natury odzwierciedlają podobieństwa, doskonale się nadają do tego celu.

Nowoczesne modele LLM szeroko wykorzystują osadzenia, które zazwyczaj mają ok. 512 wymiarów i zapewniają wierną reprezentację danych językowych. Dzięki dużej liczbie wymiarów modele te są w stanie rozpoznawać szeroką gamę skomplikowanych wzorców i doskonale sobie radzą w takich zastosowaniach jak tłumaczenie i streszczanie tekstów czy generowanie odpowiedzi wiernie oddających naturalny dyskurs.

Jeżeli dwa teksty zawierają podobne treści, ich reprezentacje wektorowe również są podobne. Rysunek 2.8 przedstawia trzy zdania w dwuwymiarowej przestrzeni osadzeń. Zdania „Kot goni mysz po całym domu" i „Mysz po całym domu ucieka przed kotem" różnią się składnią, ale zawierają tę samą treść, więc ich osadzenia są podobne. Natomiast temat zdania „Astronauta naprawia statek kosmiczny na orbicie" (astronauta i statek kosmiczny) nie jest w żaden sposób powiązany z tematami poprzednich zdań (kot i mysz), więc jego osadzenie jest zupełnie inne. W tym przykładzie, dla przejrzystości, osadzenie ma tylko dwa wymiary. W praktyce jest ich znacznie więcej, na przykład 512.

Rysunek 2.8. Trzy zdania w dwuwymiarowej przestrzeni osadzeń

W następnych rozdziałach wielokrotnie będziemy się odwoływać do interfejsów API osadzeń, ponieważ odgrywają one istotną rolę w przetwarzaniu języka naturalnego za pomocą modeli sztucznej inteligencji.

Modele moderujące

Jak wspomniano wcześniej, modeli OpenAI należy używać zgodnie z określonymi zasadami (*https://openai.com/policies/usage-policies*). Laboratorium OpenAI udostępnia model sprawdzający, czy zadana treść spełnia te wymogi, przydatny w aplikacjach, w których monitami są dane wprowadzane przez użytkowników. Na podstawie wyników moderacji można odrzucać niedopuszczalne treści. Model klasyfikuje dane wejściowe według następujących kategorii:

Nietolerancja

Nietolerancja ze względu na rasę, płeć, pochodzenie etniczne, religię, narodowość, orientację seksualną, niepełnosprawność lub przynależność do kast.

Groźby

Nienawiść, przemoc, wyrządzanie krzywdy.

Autoagresja

Samookaleczenia, samobójstwa, uszkodzenia ciała, zaburzenia odżywiania.

Seks

Aktywność seksualna, usługi seksualne. Wyjątkiem są treści edukacyjne i zdrowotne.

Seks z nieletnimi

Treści o charakterze jednoznacznie seksualnym z udziałem osób poniżej 18 roku życia.

Przemoc

Przemoc, zadawanie cierpienia, poniżanie.

Drastyczna przemoc

Drastyczne treści przedstawiające śmierć, przemoc lub poważne obrażenia ciała.

Klasyfikacja treści w językach innych niż angielski jest ograniczona.

Punkt końcowy modelu moderującego to `openai.Moderation`. Jego metoda `create()` ma tylko dwa parametry: `model` i `input`. Istnieją dwa modele moderujące. Domyślny, `text-moderation-latest`, jest na bieżąco aktualizowany, a więc zawsze najdokładniejszy. Drugi to `text-moderation-stable`. Laboratorium OpenAI powiadamia o jego planowanych aktualizacjach.

Model `text-moderation-stable` jest nieco mniej dokładny niż `text-moderation-latest`.

Poniższy listing przedstawia przykład użycia modelu moderującego:

```
import openai
# Odwołanie do punktu końcowego openai.Moderation z użyciem modelu text-moderation-latest
response = openai.Moderation.create(
    model="text-moderation-latest",
    input="Chcę zabić sąsiada.",
)
```

Obiekt `response` zawiera następującą odpowiedź:

```
{
  "id": "modr-8HTbry9SKJEjgrzDTOwA3BYaICe4k",
  "model": "text-moderation-006",
  "results": [
```

```
      {
        "flagged": true,
        "categories": {
          "sexual": false,
          "hate": true,
          "harassment": true,
          "self-harm": false,
          "sexual/minors": false,
          "hate/threatening": true,
          "violence/graphic": false,
          "self-harm/intent": false,
          "self-harm/instructions": false,
          "harassment/threatening": true,
          "violence": true
        },
        "category_scores": {
          "sexual": 1.7157400407086243e-06,
          "hate": 0.9228031039237976,
          "harassment": 0.9820413589477539,
          "self-harm": 9.009458153741434e-06,
          "sexual/minors": 6.114101779530756e-06,
          "hate/threatening": 0.346671462059021,
          "violence/graphic": 3.973896980369318e-07,
          "self-harm/intent": 2.4827872380228655e-07,
          "self-harm/instructions": 6.589202961038154e-09,
          "harassment/threatening": 0.9882690906524658,
          "violence": 0.9987002611160278
        }
      }
    ]
}
```


Tabela 2.7 opisuje informacje zawarte w odpowiedzi moderującego punktu końcowego.

Laboratorium OpenAI stale udoskonala model moderujący. W efekcie mogą się zmieniać zarówno oceny `category_scores`, jak i wartości progowe determinujące przynależność treści do poszczególnych kategorii.

Whisper i DALL-E

Laboratorium OpenAI udostępnia kilka narzędzi, które nie są modelami LLM, ale w określonych sytuacjach można je stosować w połączeniu z modelami GPT. Nie opisujemy ich tutaj szczegółowo, ponieważ wykraczają poza zakres niniejszej książki. Niemniej z ich interfejsów API korzysta się bardzo podobnie jak w przypadku modeli LLM.

Tabela 2.7. Opis odpowiedzi moderującego punktu końcowego

Atrybut	Typ	Opis
`model`	Ciąg znaków	Użyty model moderujący. W odwołaniu do punktu końcowego został wskazany model `model text-moderation-latest`, natomiast wynik zawiera `text-moderation-006`. W przypadku użycia modelu `text-moderation-stable` wynik zawierałby model `text-moderation-005`.
`flagged`	Wartość logiczna	Wartość `true`, jeżeli model sklasyfikuje treść jako niezgodną z zasadami OpenAI, lub `false` w przeciwnym wypadku.
`categories`	Słownik	Słownik wartości logicznych określających kategorie treści. Wartość `true` oznacza, że treść należy do danej kategorii naruszającej zasady, a `false`, że nie należy. Słownik można wyświetlić za pomocą polecenia `print(type(response['results'][0]['categories'])).`
`category_scores`	Słownik	Słownik ocen z przedziału od 0 do 1 przynależności treści do poszczególnych kategorii naruszających zasady OpenAI. Im wyższa ocena, tym większa pewność, że treść należy do danej kategorii. Oceny nie są wartościami prawdopodobieństwa. Słownik można wyświetlić za pomocą polecenia `print(type(response['results'][0]['category_scores'])).`

Whisper to wszechstronny, wielozadaniowy model do rozpoznawania mowy, wytrenowany na dużym zbiorze danych dźwiękowych. Potrafi identyfikować język, rozpoznawać mowę w różnych językach i ją tłumaczyć. Projekt modelu jest otwarty i dostępny w serwisie GitHub (*https://github.com/openai/whisper*).

W styczniu 2021 r. laboratorium OpenAI udostępniło DALL-E — model tworzący realistyczne obrazy na podstawie opisów w języku naturalnym. Jego wariant DALL-E 2 wykorzystuje w jeszcze większym stopniu technologie przetwarzania tekstu i nowe możliwości. Oba warianty są transformerami wytrenowanymi na obrazach i ich tekstowych opisach. Model DALL-E 2 można wypróbować za pomocą jego interfejsu API lub na stronie *https://labs.openai.com*.

Podsumowanie (i ściągawka)

Jak się przekonałeś, laboratorium OpenAI udostępnia swoje modele jako usługi za pośrednictwem interfejsów API. W tej książce wykorzystujemy bibliotekę Pythona, opracowaną przez OpenAI, będącą prostą obudową dla interfejsów. Za jej pomocą można implementować interakcje z modelami GPT-4 i ChatGPT. To pierwszy krok do tworzenia aplikacji opartych na modelach LLM! Jednak podczas korzystania z modeli pamiętaj o zarządzaniu kluczami API, kosztach i zasadach prywatności danych.

Przed rozpoczęciem zalecamy przeczytanie zasad korzystania z usług OpenAI i wypróbowanie platformy Playground, za pomocą której można zapoznać się z modelami bez kłopotliwego kodowania. Pamiętaj: najlepszym wyborem jest model GPT-3.5 Turbo, bazujący na ChatGPT.

Aby wysłać dane do modelu GPT-3.5 Turbo, skorzystaj z poniższej ściągawki:

1. Zainstaluj moduł `openai`:

   ```
   pip install openai
   ```

2. Zapisz klucz API w zmiennej środowiskowej:

   ```
   export OPENAI_API_KEY=sk-...
   ```

3. W kodzie w języku Python zaimportuj moduł `openai`:

   ```
   import openai
   ```

4. Wywołaj punkt końcowy `openai.ChatCompletion`:

   ```
   response = openai.ChatCompletion.create(
       model="gpt-3.5-turbo",
       messages=[{"role": "user", "content": "Tekst wejściowy"}],
   )
   ```

5. Wyodrębnij odpowiedź:

   ```
   print(response['choices'][0]['message']['content'])
   ```

Pamiętaj o sprawdzaniu kosztów na stronie *https://openai.com/pricing* i szacowaniu ich za pomocą biblioteki *tiktoken* (*https://github.com/openai/tiktoken*).

Pamiętaj, aby nie wysyłać do punktów końcowych poufnych danych, takich jak dane osobowe i hasła.

Laboratorium OpenAI oferuje jeszcze kilka innych modeli i narzędzi. W następnych rozdziałach poznasz bardzo przydatny punkt końcowy umożliwiający wykorzystywanie osadzeń w aplikacjach przetwarzających język naturalny.

Teraz, gdy wiesz już, jak korzystać z usług OpenAI, nadszedł czas, aby dowiedzieć się, *dlaczego* warto to robić. W następnym rozdziale znajdziesz przykłady i opisy zastosowań, które pozwolą Ci w pełni wykorzystać modele ChatGPT i GPT-4.


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

Autorzy wytyczają ścieżkę do tworzenia najnowocześniejszych aplikacji!

Tom Taulli, autor Generative AI

ChatGPT wywołał wstrząs w branży technologicznej. Programiści i wynalazcy otrzymali niesamowite możliwości dostępne na wyciągnięcie ręki. Interfejs API OpenAI i towarzyszące mu biblioteki stanowią gotowe rozwiązanie dla każdego twórcy aplikacji opartych na sztucznej inteligencji. Programista za pomocą zaledwie kilku linii kodu może implementować w swoich projektach wyrafinowane funkcje.

Ta napisana jasnym językiem książka stanowi kompleksowy przewodnik dla programistów Pythona, którzy chcą budować aplikacje bazujące na dużych modelach językowych. Zaprezentowano w niej główne cechy i zasady działania modeli GPT-4 i ChatGPT. Znalazły się tu także instrukcje, jak krok po kroku tworzyć w Pythonie aplikacje korzystające z modeli do generowania treści, odpowiadania na pytania i streszczania tekstów. Istotną zaletą są przejrzyste przykłady i dołączone pliki z kodami, pomocne w tworzeniu konkretnych projektów. Dzięki tej książce z łatwością wykorzystasz moc dużych modeli językowych w swoich aplikacjach!

Książka płynnie łączy teorię z praktyką, przystępnie opisuje zawiłości modeli GPT-4 i ChatGPT.

Lucas Soares, inżynier uczenia maszynowego w Biometrid

Dowiesz się:

- jak działają modele ChatGPT i GPT-4 i do czego mogą być przydatne
- jak korzystać z modeli NLP w aplikacjach Pythona
- jak używać interfejsów API modeli do przetwarzania języka naturalnego
- jak stosować zaawansowane techniki, takie jak inżynieria monitu
- jak dostrajać modele do określonych zadań

Dr Olivier Caelen jest badaczem i wykładowcą uniwersyteckim. Zajmuje się uczeniem głębokim. Współautor publikacji w czasopismach naukowych i współtwórca sześciu patentów.

Marie-Alice Blete jest architektem oprogramowania i inżynierem danych. Szczególnie interesuje się problemami wydajności i opóźnień we wdrażanych systemach sztucznej inteligencji.

Helion

helion.pl

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

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-289-1044-7

9 788328 910447

Cena: 59,00 zł