OKIEM EKSPERTA

Zostań ekspertem .NET 8

Dobre praktyki, wzorce projektowe, debugowanie i testowanie aplikacji

Mark J. Price

Helion

packt

Tytuł oryginału: Tools and Skills for .NET 8: Get the career you want with good practices and patterns to design, debug, and test your solutions

Tłumaczenie: Wojciech Moch

ISBN: 978-83-289-2233-4

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

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

Rozdział 9

Tworzenie chatu używającego modelu LLM

Integracja **wielkiego modelu językowego** (ang. *Large Language Model* — LLM), takiego jak GPT-4o firmy OpenAI, z projektem .NET może znacząco zwiększyć jego możliwości, oferując rozbudowane funkcje przetwarzania języka naturalnego, generowania treści itd.

Ten rozdział został poświęcony budowie usługi chatu wykorzystującej model LLM, która dostosuje standardowy model do własnych potrzeb poprzez uzupełnienie go o niestandardowe informacje, w tym biografię autora tej książki oraz dane z bazy Northwind. Dzięki temu użytkownik będzie mógł zadawać pytania dotyczące autora oraz bazy danych fikcyjnej korporacji.

Wskazówka

W momencie pisania tego rozdziału usługi API chmurowych modeli LLM są płatne. W tym rozdziale skorzystamy z jednej z najtańszych opcji: GPT-3.5 Turbo firmy OpenAI. Jak podaje ona na swoich stronach: „GPT-3.5 Turbo to nasz szybki i tani model do prostszych zadań. Model *gpt-3.5-turbo-0125* jest flagowym modelem tej rodziny — obsługuje kontekst o wielkości 16K i jest zoptymalizowany na potrzeby prowadzenia dialogów". Koszt danych wejściowych wynosi 0,50 USD za 1 mln tokenów, a koszt danych wyjściowych to 1,50 USD za 1 mln tokenów. Wykonanie zadań z tego rozdziału kosztowało mnie mniej niż 0,03 USD.

Można też użyć modelu GPT-4o, ponieważ: „GPT-4o to nasz najbardziej zaawansowany model multimodalny, który jest szybszy i tańszy niż GPT-4 Turbo, a także ma większe możliwości przetwarzania obrazów. Model obsługuje kontekst o wielkości 128K i jest wyposażony w bazę wiedzy aktualną na październik 2023 r.". Koszt danych wejściowych wynosi 5,00 USD za 1 mln tokenów, a koszt danych wyjściowych to 15,00 USD za 1 mln tokenów.

W tym rozdziale zajmiemy się następującymi zagadnieniami:

- wprowadzenie do modeli LLM,
- korzystanie z funkcji Semantic Kernel z modelem OpenAI,
- uruchamianie lokalnych modeli LLM.

Wprowadzenie do modeli LLM

Pojęcia **wielki model językowy** (LLM) i **generatywny, wstępnie wytrenowany transformator** (ang. *Generative Pre-trained Transformer* — GPT) są często używane zamiennie, ale istnieją między nimi subtelne różnice:

- LLM to bardziej ogólne określenie, odnoszące się do każdego modelu językowego z dużą liczbą parametrów (liczoną w miliardach), który został wytrenowany na ogromnych ilościach danych tekstowych i potrafi wykonywać różnorodne zadania językowe, takie jak tłumaczenia, streszczanie, odpowiadanie na pytania i inne.
- GPT to seria modeli opracowanych przez firmę OpenAI. Można to porównać do różnicy między iPhone'em a smartfonami — iPhone to rodzaj smartfona. Podobnie GPT jest rodzajem LLM. „Transformator" w nazwie opisuje architekturę, na której są zbudowane wszystkie modele rodziny GPT. Jest to przełomowa struktura modelu wprowadzona w 2017 r., która szczególnie dobrze radzi sobie ze sekwencjami danych takimi jak zdania w tekście. Chociaż wszystkie modele GPT bazują na architekturze transformatora, nie oznacza to, że robią to wszystkie modele LLM.

Zatem każdy model GPT jest modelem LLM, ale nie każdy model LLM jest modelem GPT.

Jak działają modele LLM?

Modele LLM działają na zasadach uczenia maszynowego, w szczególności wykorzystując techniki głębokiego uczenia. Modele LLM są trenowane na ogromnych zbiorach danych tekstowych pochodzących z książek, stron internetowych takich jak Stack Overflow i Reddit, artykułów oraz innych treści tekstowych. Dzięki temu dane te zapewniają modelowi szeroką znajomość języka, kontekstu i wiedzy faktograficznej aż do momentu zakończenia treningu.

Uwaga

Więcej informacji o partnerstwie OpenAI ze Stack Overflow znajdziesz pod tym adresem: *https://stackoverflow.co/company/press/archive/openai-partnership*. Szczegóły dotyczące partnerstwa OpenAI z Redditem dostępne są tutaj: *https://openai.com/index/openai-and-reddit-partnership*.

Modele LLM wykorzystują typ architektury sieci neuronowych nazywanej **transformatorem**. Architektura ta składa się z wielu warstw mechanizmów uwagi i sieci typu *feed-forward*. Do jej kluczowych elementów należą:

- **Mechanizmy uwagi.** Pozwalają modelowi ważyć znaczenie różnych słów w zdaniu, pomagając skupić się na istotnych częściach wejścia przy generowaniu odpowiedzi.
- **Sieci typu *feed-forward*.** Przetwarzają wejścia z mechanizmów uwagi i generują wyniki na każdym poziomie przetwarzania.

Uwaga

Obecna „ograniczona inteligencja" modeli LLM wynika z ich jednokierunkowej natury (ang. *feed-forward-only*). Eksperymenty z systemami wykorzystującymi sprzężenie zwrotne mogą doprowadzić do powstania systemów o wyższym poziomie rozpoznawania informacji, umożliwiając autorefleksję i ciągłe uczenie się.

Wejściowy tekst jest dzielony na mniejsze jednostki zwane tokenami. Tokeny mogą być słowami, częściami słów lub nawet pojedynczymi znakami. Tokenizacja pomaga modelowi efektywnie obsługiwać duże zbiory słów. Na przykład zdanie „Witaj, świecie!" może zostać podzielone na takie tokeny: `["Witaj", ",", "świecie", "!"]`.

Proces trenowania składa się z następujących kroków:

- **Przetwarzanie w przód** (ang. *forward pass*). Tekst wejściowy przechodzi przez model, który generuje najbardziej prawdopodobne wyjście (np. przewiduje następne słowo).
- **Obliczanie straty.** Przewidywanie modelu jest porównywane z rzeczywistym kolejnym słowem z danych treningowych i obliczana jest wartość straty. Strata mierzy różnicę między przewidywaniem a rzeczywistym słowem.
- **Przetwarzanie wstecz i optymalizacja.** Wykorzystując techniki takie jak wsteczna propagacja błędu i gradientowy spadek, model dostosowuje swoje wewnętrzne parametry, aby zminimalizować stratę. Proces ten powtarza się wiele razy na zbiorze danych treningowych.

Po początkowym treningu modele są często dostrajane na specyficznych zbiorach danych, aby poprawić wydajność w określonych dziedzinach lub zadaniach. Dostrajanie (ang. *fine-tuning*) polega na dalszym trenowaniu modelu na mniejszym, bardziej ukierunkowanym zbiorze danych. Na przykład wydawnictwo mogłoby dostroić własny model LLM za pomocą treści wszystkich swoich opublikowanych książek i wykorzystać go do obsługi chatbotów w swoich kanałach kontaktu z klientami. Chatbot mógłby wtedy odpowiadać na pytania dotyczące książek wydawnictwa i pomagać czytelnikom w razie problemów z konkretnymi stronami lub rozdziałami.

Podczas wnioskowania (czyli użytkowania) przeszkolony model generuje tekst na podstawie otrzymanego wejścia. W tym celu musi wykorzystać:

- **Rozumienie kontekstu.** Model wykorzystuje swoją wytrenowaną wiedzę, aby zrozumieć kontekst wejściowego tekstu.
- **Przewidywanie.** Model przewiduje najbardziej prawdopodobny następny token lub sekwencję tokenów, generując spójny tekst odpowiedni dla danego wejścia.

Modele LLM potrafią radzić sobie z niejednoznacznością i utrzymywać kontekst w dłuższych fragmentach tekstu dzięki zastosowaniu mechanizmów uwagi. Potrafią przypominać sobie istotne informacje z wcześniejszych części rozmowy, co pozwala im generować spójne i kontekstowo trafne odpowiedzi. Używane w modelach okna kontekstowe są nieustannie rozwijane.

Oprócz ogromnych możliwości modele LLM mają też pewne ograniczenia:

- **Stronniczość.** Modele mogą dziedziczyć uprzedzenia obecne w danych treningowych, w efekcie czego generują stronnicze lub niesprawiedliwe odpowiedzi.
- **Nieścisłości faktograficzne.** Modele generują tekst podobny do ludzkich wypowiedzi, mogą również tworzyć niepoprawne lub nonsensowne informacje. Jest to szczególnie niebezpieczne, ponieważ tworzą realistycznie brzmiące porady, np. prawne. Modele LLM należy zatem traktować jak młodszych członków zespołu, których praca zawsze wymaga weryfikacji.
- **Zależność od danych.** Ich wiedza jest ograniczona do danych, na których zostały wytrenowane, a te mogą być nieaktualne w czasie, gdy używasz modelu.

Modele LLM znajdują zastosowanie w różnych dziedzinach, np.:

- **Chatboty i wirtualni asystenci.** Zapewniają użytkownikom poziom interakcji przypominający rozmowę z prawdziwym człowiekiem, np. ChatGPT-4 czy Microsoft Copilot.
- **Tworzenie treści.** Udzielają pomocy w pisaniu artykułów, raportów, a nawet tekstów kreatywnych, np. Microsoft Designer: *https://designer.microsoft.com/*.
- **Tłumaczenia.** Pozwalają na oferowanie usług tłumaczenia tekstów.
- **Pomoc przy programowaniu.** Udzielają pomocy programistom, sugerując kierunek tworzenia kodu i wspomagając przy debugowaniu, np. GitHub Copilot i JetBrains AI Assistant.

Skoro już wiemy, jak działają LLM i jakie mają zalety, dowiedzmy się, jak można uzyskać do nich dostęp.

Uzyskiwanie dostępu do modelu LLM

Na potrzeby naszego małego chatu będziemy potrzebować dostępu do modelu LLM. Najprostszym sposobem uzyskania dostępu do chmurowego modelu LLM przystosowanego do integracji z projektami .NET jest skorzystanie z usług OpenAI lub Azure OpenAI. W momencie pisania tego tekstu aplikowanie o model Azure OpenAI wymaga posiadania adresu e-mail w domenie firmowej. Konta Gmail lub MSN nie są akceptowane. Dlatego w tym rozdziale będziemy korzystać głównie z modeli firmy OpenAI.

> **Uwaga**
>
> Jeżeli masz adres e-mail w domenie firmowej i wolisz korzystać z płatnej usługi Azure, to pod tym adresem dowiesz się, jak przygotować dla siebie model Azure OpenAI: *https://learn.microsoft.com/azure/ai-services/openai/how-to/create-resource*.

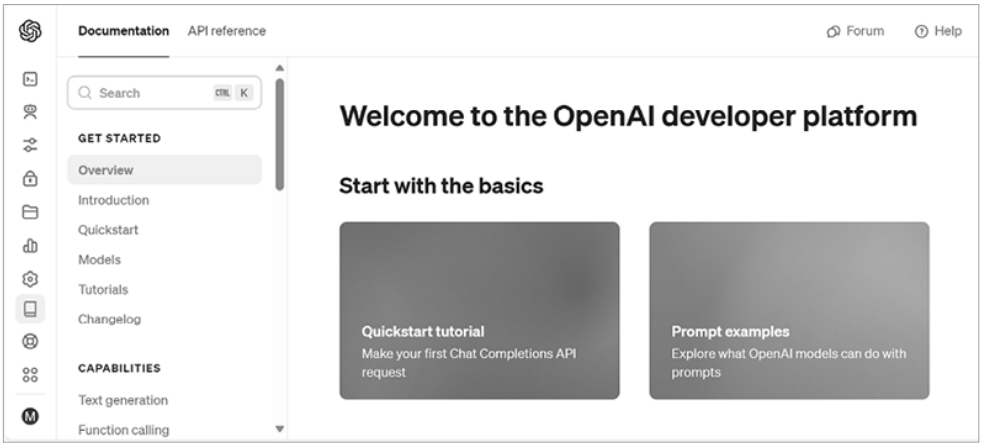
Aby skorzystać z chmurowego modelu LLM, będą nam potrzebne dwie informacje:

- **Klucz API.** Udostępniany jest w portalu usługi LLM i powinien być przechowywany w bezpiecznym miejscu. Każdy, kto posiada Twój klucz API, może korzystać z zasobów Twojego konta.

- **Identyfikator modelu chatu.** W naszym przypadku użyjemy modelu *gpt-3.5-turbo*, który jest dobrą kombinacją niskich kosztów i rozsądnego poziomu inteligencji. Więcej o tym modelu można przeczytać pod adresem *https://platform.openai.com/docs/models/gpt-3-5-turbo*.

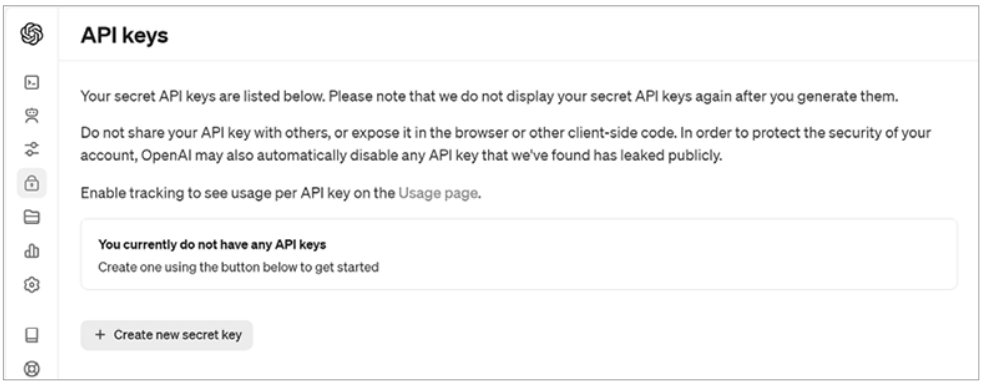
No to zaczynamy:

1. Otwórz przeglądarkę i przejdź na stronę rejestracji firmy OpenAI pod adresem *https://platform.openai.com/signup*.
2. Po zarejestrowaniu się i zalogowaniu zobaczysz stronę główną platformy deweloperskiej OpenAI, taką jak na rysunku 9.1.



Rysunek 9.1. Strona główna platformy deweloperskiej OpenAI

3. W menu nawigacyjnym po lewej stronie przejdź do sekcji *API keys* lub przejdź bezpośrednio pod adres *https://platform.openai.com/api-keys*. Kliknij tutaj przycisk *+Create new secret key*, tak jak na rysunku 9.2.



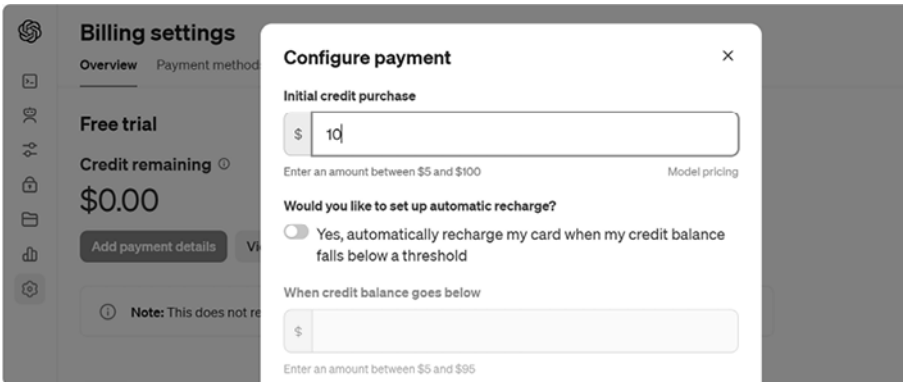
Rysunek 9.2. Strona kluczy dostępu do API OpenAI

4. Wprowadź nazwę, np. *tools-skills-net8*, a następnie kliknij przycisk *Create secret key*.
5. Gdy pojawi się okno dialogowe *Save your key*, skopiuj klucz do schowka i zapisz go w bezpiecznym i dostępnym miejscu.

Dobra praktyka

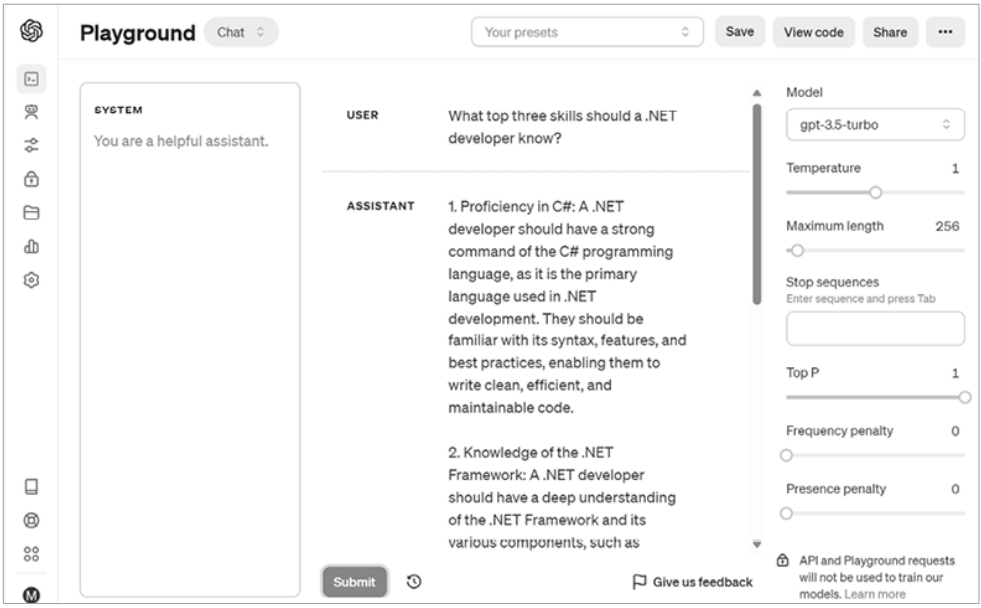
Zapisz swój klucz w bezpiecznym miejscu, ponieważ nie ma możliwości ponownego wyświetlenia go w koncie OpenAI. Jeżeli zgubisz klucz, trzeba będzie wygenerować nowy.

6. W menu nawigacyjnym po lewej stronie przejdź do pozycji *Settings/Billing* i kliknij przycisk *Payment methods*, a następnie dodaj nową metodę płatności, np. kartę Visa, lub przejdź pod adres *https://platform.openai.com/account/billing/overview*.
7. Na stronie konfigurowania rozliczeń w zakładce *Overview* kliknij przycisk *Add to credit balance* i dodaj środki. Zazwyczaj wydasz jedynie kilka centów, więc zalecam dodanie minimalnej kwoty. Ja dodałem 10 USD, ale możesz dodać minimalną wartość, tak jak na rysunku 9.3.



Rysunek 9.3. Wpłacanie pieniędzy na konto OpenAI

8. W menu nawigacyjnym po lewej stronie przejdź do sekcji *Playground* i upewnij się, że na liście rozwijanej obok tytułu strony wybrana jest wartość *Chat*, a z listy modeli wybrana jest wartość *gpt-3.5-turbo*. Możesz także przejść pod adres *https://platform.openai.com/playground?model=gpt-3.5-turbo*. Oczywiście można też dokładniej wybrać numer wersji modelu, np. *gpt-3.5-turbo-0125*, ale bardziej ogólna nazwa oznacza, że zawsze będzie używana najnowsza wersja.
9. Aby przetestować dostępność swoich środków, wpisz następujące pytanie użytkownika: **`Jakie trzy najważniejsze umiejętności powinien znać programista .NET?`**, kliknij przycisk *Submit*, a następnie przyjrzyj się wynikowi, który pokazano na rysunku 9.4.



Rysunek 9.4. Testowanie dostępnych środków w piaskownicy

Ostrzeżenie!

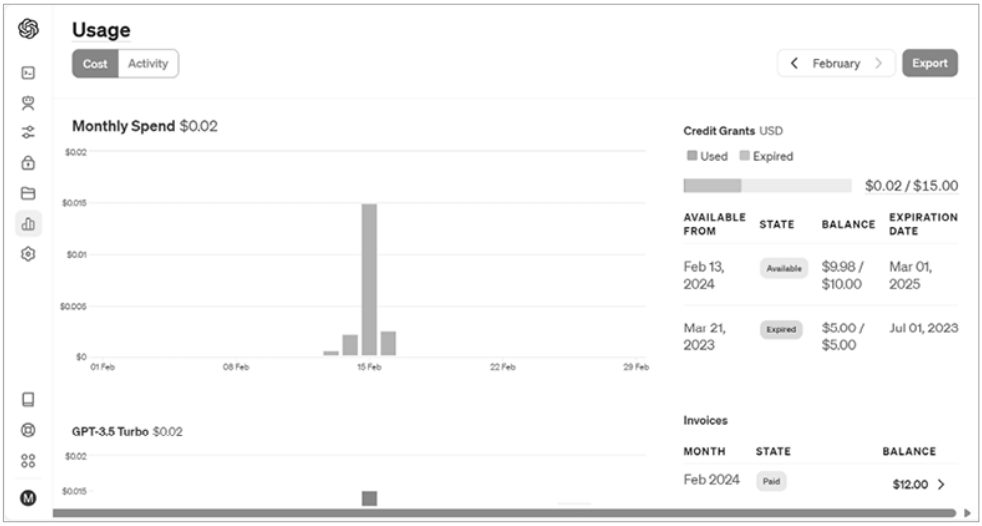
Jeżeli pojawi się błąd mówiący o braku środków, to odczekaj kilka minut, aby system zorientował się, że przed chwilą wpłacono jakąś kwotę. Jednym z powodów, dla których używamy teraz usługi Playground, jest sprawdzenie, czy wpłacone środki rzeczywiście zostały przyjęte przez systemy OpenAI.

10. W menu nawigacyjnym po lewej stronie wybierz pozycję *Usage* lub przejdź pod adres *https://platform.openai.com/usage*, gdzie można się dowiedzieć, że używane tutaj wywołania API powinny kosztować tylko kilka centów, tak jak na rysunku 9.5.
11. Zamknij przeglądarkę.

Teraz możemy już przystąpić do integrowania modelu LLM z naszym projektem .NET.

Używanie Semantic Kernel z modelem OpenAI

Eksperci AI tworzący wielkie modele językowe (LLM) zazwyczaj używają systemu operacyjnego Linux i języka Python. Samouczki i dokumentacja często zakładają, że deweloperzy również będą korzystać z tych platform. Użycie innego języka programowania lub



Rysunek 9.5. Wykresy obrazujące wykorzystanie API

frameworka, który nie został zaprojektowany dla platformy .NET, może wprowadzać dodatkową złożoność i zmniejszać łatwość utrzymania projektu. Poza tym rozwiązywanie problemów wynikających z różnic między językami lub frameworkami to dodatkowe kłopoty w pracy.

Czym jest Semantic Kernel?

Semantic Kernel został zaprojektowany z myślą o ekosystemie .NET, zapewniając integrację z językiem C# i innymi językami .NET. Stanowi to ułatwienie w pracy dla programistów .NET, ponieważ otrzymują oni biblioteki lub pakiety zgodne z idiomami języka C#.

Semantic Kernel może korzystać z asynchronicznego modelu programowania w .NET do efektywniejszej realizacji zadań zależnych od wejścia-wyjścia (I/O), co jest niezwykle ważne w przypadku integracji usług wymagających komunikacji sieciowej, takich jak API modeli LLM. Wynikiem są tutaj nieblokujące operacje wejścia-wyjścia, co jest istotne dla zachowania wysokiej responsywności aplikacji.

Korzystanie z Semantic Kernel umożliwia lepszą integrację modelu w projekcie .NET. Pozwala on na precyzyjne dostosowanie sposobu, w jaki model LLM wchodzi w interakcję z aplikacją. Można np. dostosować potok żądań, sposób przetwarzania odpowiedzi lub integrację z innymi usługami .NET. Ta elastyczność umożliwia również skuteczniejsze skalowanie aplikacji, dostosowując użycie modelu LLM w miarę rozwoju projektu.

Biorąc pod uwagę popularność platformy .NET, integracja realizowana za pomocą takiego rozwiązania jak Semantic Kernel może zapewnić lepsze wsparcie ze strony społeczności i udostępnić zasoby dostosowane do potrzeb programistów .NET. Do dyspozycji mamy rozbudowaną dokumentację, fora oraz narzędzia firm trzecich, które mogą być nieocenioną pomocą podczas rozwiązywania problemów i usprawniania aplikacji.

Teraz możemy już przystąpić do budowania projektu .NET korzystającego z API OpenAI z wykorzystaniem przygotowanego wcześniej klucza.

Przede wszystkim musimy bezpiecznie przechowywać konfigurację naszej aplikacji. Użyjemy do tego funkcji obsługi konfiguracji .NET oraz kombinacji pliku ustawień JSON (będzie przechowywał nazwy modeli itd.) i zmiennych środowiskowych (tu zapiszemy nasz klucz API).

No to do roboty:

1. Użyj swojego edytora kodu, aby do rozwiązania *Rozdzial09* dodać nową aplikację konsoli o nazwie *ChatApp*.
2. Zmodyfikuj plik *ChatApp.csproj*, aby statycznie i globalnie zaimportować klasę `System.Console`, dodać odwołanie do pakietów umożliwiających współpracę z Semantic Kernel i obsługę konfiguracji przy użyciu plików JSON i zmiennych środowiskowych, włączyć traktowanie ostrzeżeń jako błędów oraz dołączyć plik *appsettings.json* z wdrożonym zestawem, zgodnie z wyróżnieniami w poniższym kodzie:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <TreatWarningsAsErrors>true</TreatWarningsAsErrors>
  </PropertyGroup>

  <ItemGroup>
    <!--Pakiety tworzenia konfiguracji z pliku appsettings.json i zmiennych środowiskowych.-->
    <PackageReference Version="8.0.1"
        Include="Microsoft.Extensions.Configuration.Binder" />
    <PackageReference Version="8.0.0"
        Include="Microsoft.Extensions.Configuration.Json" />
    <PackageReference Version="8.0.0"
        Include="Microsoft.Extensions.Configuration.EnvironmentVariables" />

    <!--Pakiety umożliwiające pracę z Semantic Kernel.-->
    <PackageReference Include="Microsoft.SemanticKernel" Version="1.13.0" />
  </ItemGroup>

  <ItemGroup>
    <Content Include="appsettings.json">
      <CopyToOutputDirectory>Always</CopyToOutputDirectory>
    </Content>
  </ItemGroup>

  <ItemGroup>
    <Using Include="System.Console" Static="true" />
  </ItemGroup>

</Project>
```

Uwaga

Najnowsze wersje pakietów Semantic Kernel znajdziesz pod tym adresem: *https://www.nuget.org/packages/Microsoft.SemanticKernel*.

3. Skompiluj projekt *ChatApp*, aby pobrać niezbędne pakiety.
4. W folderze projektu *ChatApp* dodaj nowy plik JSON o nazwie *appsettings.json* i wpisz do niego poniższy fragment:

```
{
  "Settings": {
    "ModelId": "gpt-3.5-turbo-0125",
    "OpenAISecretKey": "<twój-tajny-klucz>"
  }
}
```

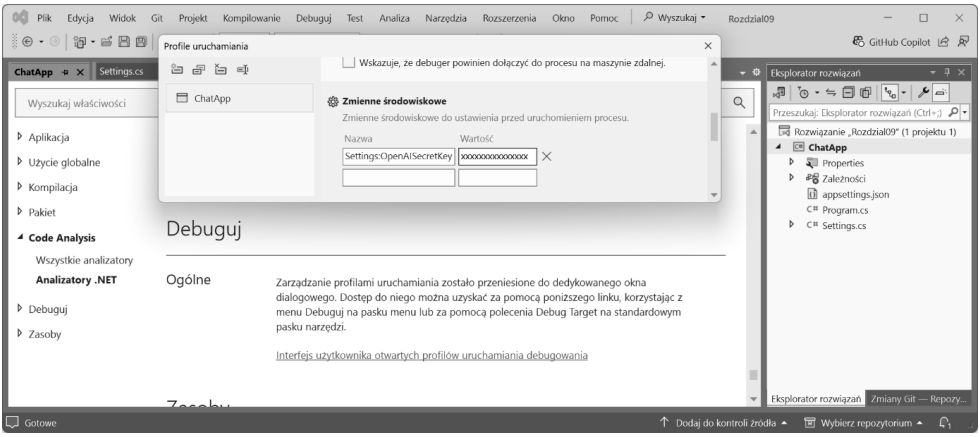
5. W folderze projektu *ChatApp* dodaj nowy plik klasy o nazwie *Settings.cs*, taki jak w poniższym kodzie:

```
public sealed class Settings
{
  public required string ModelId { get; set; }
  public required string OpenAISecretKey { get; set; }
}
```

Uwaga

Klasa `Settings` znajduje się w domyślnej przestrzeni nazw, podobnie jak automatycznie generowana klasa `Program`.

6. Jeżeli korzystasz z Visual Studio, wybierz z menu pozycję *Projekt/ChatApp właściwości*, a następnie w sekcji *Debuguj* kliknij łącze *Interfejs użytkownika otwartych profilów debugowania* i w okienku dialogowym wprowadź zmienną środowiskową, tak jak na rysunku 9.6.



Rysunek 9.6. Definiowanie zmiennej środowiskowej w profilu uruchamiania projektu

7. Zamknij okno ustawień uruchamiania i upewnij się, że utworzony został folder *Properties* zawierający plik o nazwie *launchSettings.json* z zawartością definiującą zmienną środowiskową, taką jak w poniższym kodzie:

```
{
  "profiles": {
    "ChatApp": {
      "commandName": "Project",
      "environmentVariables": {
        "Settings:OpenAISecretKey": "<twój-tajny-klucz>"
      }
    }
  }
}
```

Uwaga

Można też zdefiniować zmienną środowiskową w wierszu poleceń lub w terminalu. Jeżeli nie masz doświadczenia w tworzeniu zmiennych środowiskowych, skorzystaj z tego linku: *https://github.com/markjprice/cs12dotnet8/blob/main/docs/ch09-environment-variables.md*. Możesz także zdefiniować tajne dane użytkownika dla lokalnego środowiska deweloperskiego, a w środowisku produkcyjnym użyć usługi Azure Key Vault (lub podobnej).

8. Utwórz plik *.gitignore* w folderze projektu i dopisz do niego poniższą treść, aby upewnić się, że plik *launchSettings.json* nie będzie zapisywany w repozytorium Gita.

```
# Nie przechowuj poświadczeń w repozytorium.
launchSettings.json
**/Properties/launchSettings.json
```

Dobra praktyka

Takie pliki jak *launchSettings.json*, które mogą zawierać wrażliwe informacje, np. klucz dostępu, nigdy nie powinny być umieszczane w publicznym repozytorium Gita.

9. W folderze projektu dodaj nowy plik klasy o nazwie *Program.GetSettings.cs*.

10. W pliku *Program.GetSettings.cs* usuń istniejące instrukcje i dopisz kod definiujący metodę ładującą plik konfiguracyjny i zmienne środowiskowe, zapisującą odczytane dane w obiekcie klasy `Settings`, a następnie zwracającą ten obiekt, tak jak w poniższym kodzie:

```
// ConfigurationBuilder itd.
using Microsoft.Extensions.Configuration;

partial class Program
{
  private static Settings? GetSettings()
  {
```

```
    const string settingsFile = "appsettings.json";
    const string settingsSectionKey = nameof(Settings);

    // Tworzenie obiektu konfiguracji na bazie pliku JSON
    // i zmiennych środowiskowych.
    IConfigurationRoot config = new ConfigurationBuilder()
      .AddJsonFile(settingsFile)
      .AddEnvironmentVariables()
      .Build();

    // Pobieranie ustawień z konfiguracji na podstawie klucza
    // i silnie typowanej klasy.
    Settings? settings = config.GetRequiredSection(
      settingsSectionKey).Get<Settings>();

    if (settings is null)
    {
      WriteLine($"W pliku {settingsFile} nie znaleziono sekcji
      ↪{settingsSectionKey}.");
      return null;
    }
    else
    {
      return settings;
    }
  }
}
```

11. W folderze projektu dodaj nowy plik klasy o nazwie *Program.GetKernel.cs*.

12. W pliku *Program.GetKernel.cs* usuń wszystkie istniejące instrukcje, wpisz instrukcje tworzące obiekt typu `KernelBuilder`, dodaj funkcję interakcji z chatem OpenAI przy użyciu skonfigurowanego modelu i tajnego klucza, a następnie zwróć utworzony obiekt, tak jak w poniższym kodzie:

```
using Microsoft.SemanticKernel; // Kernel.

partial class Program
{
  private static Kernel GetKernel(Settings settings)
  {
    IKernelBuilder kernelBuilder = Kernel.CreateBuilder();

    // Konfiguruj chat OpenAI, podając nazwę modelu i swój tajny klucz.
    kernelBuilder.AddOpenAIChatCompletion(
      settings.ModelId,
      settings.OpenAISecretKey);

    Kernel kernel = kernelBuilder.Build();

    return kernel;
  }
}
```

13. W pliku *Program.cs* usuń wszystkie istniejące instrukcje i wpisz instrukcje pobierające ustawienia, używające ich do pobrania jądra semantycznego, a następnie umożliwiające chatowanie w pętli, którą można zakończyć naciśnięciem klawisza *X*, tak jak w poniższym kodzie:

```
using Microsoft.SemanticKernel; // Kernel.

Settings? settings = GetSettings();
if (settings is null)
{
  WriteLine("Nie znaleziono ustawień lub ustawienia są nieprawidłowe.
  ↪Aplikacja zostanie zamknięta.");
  return; // Zamknij aplikację.
}

Kernel kernel = GetKernel(settings);

ConsoleKey key = ConsoleKey.A;

while (key is not ConsoleKey.X)
{
  Write("Wpisz swoje pytanie: ");
  string question = ReadLine()!;
  WriteLine(await kernel.InvokePromptAsync(question));
  WriteLine();
  WriteLine("Naciśnij klawisz X, aby zakończyć, albo inny klawisz,
  ↪aby zadać kolejne pytanie.");
  key = ReadKey(intercept: true).Key;
}
```

14. Uruchom projekt *ChatApp* bez debugowania.

15. Wpisz takie pytanie: „Jakie trzy najważniejsze umiejętności powinien mieć programista .NET?”. Przyjrzyj się odpowiedzi (oczywiście może być inna, ponieważ istnieje w tym pewien element losowości), którą pokazano w poniższym przykładzie:

```
Wpisz swoje pytanie: Jakie trzy najważniejsze umiejętności powinien mieć
programista .NET?
1. Znajomość języków programowania .NET (C# lub VB.NET). Biegłość w C#,
w tym nowoczesne funkcje jak LINQ, async/await oraz pisanie czystego
I czytelnego kodu.

2. Znajomość ekosystemu .NET i technologii pokrewnych. Zrozumienie .NET
Framework, .NET Core/.NET 6+, ASP.NET Core (MVC, Web API, Blazor) oraz
pracy z bazami danych (Entity Framework Core).
3. Umiejętność projektowania i stosowania wzorców architektonicznych.
Znajomość wzorców projektowych (np. Dependency Injection), API (REST,
GraphQL) i architektur, takich jak mikroserwisy czy DDD.

Naciśnij klawisz X, aby zakończyć, albo inny klawisz, aby zadać kolejne
pytanie.
```

16. Naciśnij dowolny klawisz, a następnie wprowadź kolejne pytanie: „Proszę, napisz krótką biografię Marka Price'a". Przyjrzyj się odpowiedzi, którą pokazano na poniższym wydruku:

```
Mark Price to były amerykański koszykarz, który zyskał sławę jako jeden
z najlepszych rozgrywających NBA w latach 80. i 90. Urodzony 15 lutego
1964 roku w Bartlesville, Oklahoma, był czterokrotnym uczestnikiem Meczu
Gwiazd NBA. Price spędził większość swojej kariery w Cleveland Cavaliers,
gdzie był znany z precyzyjnego rzutu, szczególnie z rzutów wolnych,
osiągając jedne z najwyższych procentów skuteczności w historii ligi.
Po zakończeniu kariery zawodniczej zajął się trenowaniem, zarówno na
poziomie uniwersyteckim, jak i w NBA. Mark Price jest uważany za jednego
z najlepszych strzelców i najinteligentniejszych graczy swoich czasów.
```

Uwaga

Najsłynniejszym Markiem Price'em w internecie jest koszykarz.

17. Naciśnij klawisz *X*, aby zakończyć.

Praktyczne aplikacje chatbotów AI zazwyczaj muszą pracować z danymi na temat naszej organizacji, a te zazwyczaj zapisane są w relacyjnej bazie danych lub innym magazynie danych.

Przyjrzyjmy się teraz funkcjom, które mogą dostarczyć naszemu chatbotowi uzupełniające informacje, aby mógł lepiej spełniać przypisane mu zadania.

Czym są funkcje?

W kontekście modeli OpenAI funkcje odnoszą się do pewnych cech lub operacji, które te modele mogą wykonywać na żądanie. Takie funkcje rozszerzają możliwości modeli językowych poza samo generowanie tekstu, umożliwiając im realizację bardziej ustrukturyzowanych zadań.

Funkcje w modelach OpenAI to z góry zdefiniowane operacje lub zadania, które model może wykonać na żądanie. Funkcje pozwalają modelowi wykonywać wyspecjalizowane działania na podstawie otrzymanych danych wejściowych, co zwiększa jego użyteczność i elastyczność w wybranych zastosowaniach.

Funkcje są bardzo zróżnicowane w zależności od zastosowania. Oto kilka typowych przykładów:

- **Manipulacje na tekstach.** Chodzi o takie działania jak streszczanie tekstu, tłumaczenie między językami czy wyodrębnianie określonych informacji z podanego tekstu.
- **Analiza danych.** Wykonywanie obliczeń, generowanie wykresów, analizowanie zbiorów danych czy wyszukiwanie wzorców w podanych danych.
- **Pomoc w programowaniu.** Funkcje pomagające w pisaniu, debugowaniu i rozumieniu kodu w różnych językach programowania.

- **Interakcja z interfejsami API.** Funkcje zaprojektowane do interakcji z zewnętrznymi systemami, bazami danych lub API w celu pobierania lub wysyłania informacji. Jest to typ funkcji, którego używamy w celu rozbudowania modelu OpenAI o niestandardowe informacje.

Funkcje zazwyczaj działają zgodnie z ustaloną strukturą procesu:

- **Wywołanie.** Funkcja jest wywoływana lub uruchamiana na podstawie określonych słów kluczowych lub instrukcji zawartych w danych wejściowych użytkownika. Może to być wywołanie jawne (bezpośrednie wywołanie funkcji) lub niejawne (model rozpoznaje zadanie, które odpowiada danej funkcji).
- **Przetwarzanie.** Po wywołaniu funkcja przetwarza dane wejściowe zgodnie z określoną logiką lub algorytmem. Może to obejmować analizowanie tekstu, wykonywanie obliczeń, zapytania do baz danych lub interakcje z API.
- **Generowanie wyniku.** Po przetworzeniu otrzymanych danych funkcja generuje wynik, który jest następnie zwracany do modelu, aby ten mógł go przetworzyć przed zaprezentowaniem użytkownikowi. Wynik ten może mieć formę odpowiedzi tekstowej, struktury danych lub mieć zupełnie inny format.

Do korzyści wynikających z używania funkcji można zaliczyć:

- Upraszczanie złożonych zadań przez generowanie szybkich i precyzyjnych odpowiedzi na konkretne zapytania.
- Umożliwienie modelowi wykonywania wyspecjalizowanych zadań, które wymagają czegoś więcej niż tylko ogólne rozumienie języka.
- Integrację z zewnętrznymi systemami, bazami danych i API, dzięki czemu model jest bardziej wszechstronnym narzędziem, przydatnym w szerokim zakresie zastosowań.

Funkcje stanowią potężne rozszerzenie możliwości modeli OpenAI, umożliwiając im realizację różnorakich zadań i efektywną interakcję z zewnętrznymi systemami.

Teraz dodajmy do naszej aplikacji funkcję, aby dostarczyć chatbotowi więcej informacji o mnie i o bazie danych Northwind.

Dodawanie funkcji

Istnieją dwa podstawowe sposoby użycia funkcji w połączeniu z modelem OpenAI. Po pierwsze, można zdefiniować własną funkcję i bezpośrednio ją wywołać. Po drugie, można skonfigurować wiele funkcji i umożliwić interfejsowi API podjęcie decyzji, kiedy ich używać, co model zrobi, wykorzystując opisy tekstowe i zapytania użytkownika. Przeanalizujemy tutaj oba rozwiązania: najpierw pojedynczą funkcję do zadawania pytań o autora, a następnie dowolną funkcję zarejestrowaną we wtyczce.

Do roboty!

1. W pliku *ChatApp.csproj* dodaj odwołanie do projektu biblioteki klas kontekstu danych Northwind, którą przygotowaliśmy w rozdziale 1. Możesz do tego wykorzystać poniższy kod:

```
<ItemGroup>
  <ProjectReference
Include="..\..\Rozdzial01\Northwind.DataContext\Northwind.DataContext.
↪csproj" />
</ItemGroup>
```

2. Skompiluj projekt *ChatApp*, aby skompilować też dołączany projekt i skopiować utworzony plik zestawu do lokalnego folderu *bin*. Możesz też użyć polecenia `dotnet build`.

3. Utwórz nowy plik klasy o nazwie *Program.ChatFunctions.cs*, usuń z niego istniejące instrukcje, a następnie zdefiniuj statyczne metody w częściowej klasie `Program`, które będą pozwalały na uzyskanie informacji o autorze tej książki oraz o produktach w wybranych kategoriach bazy danych Northwind. Możesz zastosować podany niżej kod:

```
using Microsoft.EntityFrameworkCore; // Include.
using Northwind.EntityModels; // NorthwindContext.
using System.Text.Json; // JsonSerializer.

partial class Program
{
  private static string GetAuthorBiography()
  {
    return """
Mark J Price to były certyfikowany trener Microsoftu (MCT) i obecny
specjalista do spraw programowania w C# oraz architekst rozwiązań dla
Azure, z ponad 20-letnim doświadczeniem w edukacji i programowaniu.
Od 1993 roku Mark zdał ponad 80 egzaminów programistycznych Microsoftu
i specjalizuje się w przygotowywaniu innych do tych egzaminów. Jego
uczniowie to zarówno profesjonaliści z ogromnym doświadczeniem, jak
i 16-letni praktykanci bez żadnego przygotowania. Mark skutecznie prowadzi
wszystkich, łącząc umiejętności edukacyjne z praktycznym doświadczeniem
w konsultingu i tworzeniu systemów dla przedsiębiorstw na całym świecie.
W latach 2001-2003 Mark pracował dla Microsoftu, pisząc oficjalne
materiały szkoleniowe w Redmond, w USA. Zespół Marka stworzył pierwsze
kursy szkoleniowe języka C# jeszcze w fazie jego wczesnej wersji alfa.
Pracując dla Microsoftu, prowadził również zajęcia typu "train-the-
trainer", aby przygotować innych trenerów MCT do pracy z C# i .NET.
W latach 2016-2022 Mark tworzył i prowadził kursy szkoleniowe dla Digital
Experience Platform firmy Optimizely, najlepszego systemu CMS budowanego
w .NET, przeznaczonego dla platform marketingu cyfrowego i e-commerce.
W 2010 roku Mark zdobył dyplom studiów podyplomowych w dziedzinie edukacji
(PGCE). Uczył matematyki na poziomie GCSE w dwóch szkołach średnich
w Londynie. Mark posiada tytuł licencjata z wyróżnieniem (BSc Hons)
w dziedzinie informatyki, uzyskany na Uniwersytecie w Bristolu
w Wielkiej Brytanii.
Mark J Price jest autorem następujących książek:
- C# 6 and .NET Core 1.0 - Modern Cross-Platform Development, 2016
- C# 7 and .NET Core - Modern Cross-Platform Development, 2017
- C# 7.1 and .NET Core 2.0 - Modern Cross-Platform Development, 2017
- C# 8.0 and .NET Core 3.0 - Modern Cross-Platform Development, 2019
```

```
- C# 9 and .NET 5 - Modern Cross-Platform Development, 2020
- C# 10 and .NET 6 - Modern Cross-Platform Development, 2021
- C# 11 and .NET 7 - Modern Cross-Platform Development Fundamentals, 2022
- Apps and Services with .NET 7, 2022
- C# 12 and .NET 8 - Modern Cross-Platform Development Fundamentals, 2023
- Apps and Services with .NET 8, 2023
""";
  }
  // Zapisanie tych informacji w pamięci podręcznej jest szybsze niż
  // odtwarzanie ich za każdym razem od nowa.
  private static JsonSerializerOptions jsonSerializerOptions =
    new() { WriteIndented = true };

  private static string GetProductsInCategory(string categoryName)
  {
    using NorthwindContext db = new();

    var products = db.Products
    .Include(p => p.Category)
    .Where(p => p.Category!.CategoryName == categoryName)
    .Select(p => new
    {
      p.ProductId,
      p.ProductName,
      p.Category!.CategoryName,
      p.UnitPrice,
      p.UnitsInStock,
      p.UnitsOnOrder,
      p.Discontinued
    })
    .ToArray();

    // Zamiana tablicy produktów na ciąg znaków JSON.
    string json = JsonSerializer.Serialize(
      products, jsonSerializerOptions);

    return json;
  }
}
```

Uwaga

Wybrano taki podzbiór pól produktów, aby ograniczyć dane przesyłane do modelu i uniknąć złożoności związanej z tabelami pokrewnymi, a mimo to umożliwić odpowiadanie na pytania dotyczące produktów i kategorii. Pominąłem tu powiązane tabele (takie jak zamówienia i dostawcy), aby zmniejszyć ilość serializowanych danych. Oznacza to, że model nie będzie umiał odpowiedzieć na takie pytania jak „Kto jest dostawcą produktu Chai?" albo „Ilu klientów zamawia produkt Chang?".

4. W pliku *Program.GetKernel.cs* w metodzie `GetKernel` przed zwróceniem obiektu `kernel` dodaj instrukcje definiujące dwie funkcje na bazie naszych metod statycznych, tak jak w poniższym kodzie:

```
// Utwórz funkcję promptu w ramach wtyczki i dodaj ją do kernela.
kernel.ImportPluginFromFunctions(pluginName:"AuthorInformation", [
  kernel.CreateFunctionFromMethod(
    method: GetAuthorBiography,
    functionName: nameof(GetAuthorBiography),
    description: "Pobiera biografię autora.")
]);

kernel.ImportPluginFromFunctions("NorthwindProducts", [
  kernel.CreateFunctionFromMethod(
    method: GetProductsInCategory,
     functionName: nameof(GetProductsInCategory),
    description: "Pobiera z bazy danych Northwind produkty danej
    ↪kategorii.")
]);
```

5. W pliku *Program.cs* po uzyskaniu obiektu `kernel` zdefiniuj funkcję na podstawie naszej metody, następnie umieść w komentarzu wywołanie metody `InvokePromptAsync` i zapisz otrzymane od użytkownika pytanie jako argument wywołania metody `InvokeAsync`, tak jak w poniższym kodzie:

```
Kernel kernel = GetKernel(settings);

// Zmienna $question zostanie zdefiniowana jako argument.
KernelFunction function = kernel.CreateFunctionFromPrompt("""
  Biografia autora: {{ authorInformation.getAuthorBiography }}.
  {{ $question }}
""");

KernelArguments arguments = new();

ConsoleKey key = ConsoleKey.A;

while (key is not ConsoleKey.X)
{
  Write("Wpisz swoje pytanie: ");
  string question = ReadLine()!;

  // WriteLine(await kernel.InvokePromptAsync(question));
  arguments["question"] = question;
  // Wywołaj pojedynczą funkcję.
  WriteLine(await function.InvokeAsync(kernel, arguments));

  WriteLine();
  WriteLine("Naciśnij klawisz X, aby zakończyć, albo inny klawisz,
  ↪żeby zadać kolejne pytanie.");
  key = ReadKey(intercept: true).Key;
}
```

6. Uruchom projekt *ChatApp* bez debugowania.
7. Wprowadź taką instrukcję: `Napisz, proszę, krótką biografię Marka Price'a` i przyjrzyj się otrzymanej odpowiedzi, która powinna wyglądać podobnie do poniższej:

```
Mark J. Price - ekspert w dziedzinie programowania z ponad 20-letnim
doświadczeniem, były Microsoft Certified Trainer (MCT) i obecny
specjalista Microsoft w zakresie programowania w C# oraz projektowania
rozwiązań w Azure.
Mark zdobył tytuł licencjata z wyróżnieniem z informatyki na Uniwersytecie
w Bristolu. W latach 2001-2003 pracował w Redmond, gdzie pisał oficjalne
materiały szkoleniowe dla Microsoftu, w tym pierwsze kursy dotyczące
języka C#. W swojej karierze zdał ponad 80 egzaminów Microsoft i pomógł
setkom osób - od początkujących po doświadczonych specjalistów - osiągnąć
podobny sukces.
Mark tworzył także kursy dla platformy Optimizely, a wcześniej nauczał
matematyki w londyńskich szkołach średnich, po zdobyciu dyplomu PGCE.
Łączy pasję do edukacji z praktycznym doświadczeniem w projektowaniu
systemów dla firm na całym świecie.
```

8. Wprowadź taką instrukcję: `Napisz biografię Marka Price'a nie dłuższą niż jeden akapit` i przyjrzyj się otrzymanej odpowiedzi, która powinna wyglądać podobnie do poniższej:

```
Mark J. Price to doświadczony programista, edukator i były Microsoft
Certified Trainer (MCT) z ponad 20-letnim doświadczeniem. Specjalizuje się
w programowaniu w C# oraz projektowaniu rozwiązań w Microsoft Azure.
Współtworzył pierwsze kursy C# w Microsoft, pracując w Redmond, a także
prowadził szkolenia dla trenerów. Jest autorem materiałów edukacyjnych
I wykładowcą, który pomaga zarówno początkującym, jak i ekspertom zdobywać
certyfikaty Microsoft. Absolwent Uniwersytetu w Bristolu, łączy wiedzę
akademicką z praktyką, tworząc rozwiązania i kursy dla firm oraz ucząc
matematyki w szkołach średnich w Londynie.
```

9. Wprowadź taką instrukcję: `Kiedy autor napisał książkę o C# 10?` i przyjrzyj się otrzymanej odpowiedzi, która powinna wyglądać podobnie do poniższej:

```
Książka "C# 10 and .NET 6 - Modern Cross-Platform Development" autorstwa
Marka J. Price'a została wydana w listopadzie 2021 roku. Jeśli
potrzebujesz dokładnej daty lub szczegółów, mogę to sprawdzić.
```

10. Wprowadź taką instrukcję: `Ile książek napisał Mark?` i przyjrzyj się otrzymanej odpowiedzi, która powinna wyglądać podobnie do poniższej:

```
Mark J Price napisał łącznie dziesięć książek.
```

11. Wprowadź taką instrukcję: `Gdzie autor nauczył się języka C#?` i przyjrzyj się otrzymanej odpowiedzi, która powinna wyglądać podobnie do poniższej:

```
Mark J. Price nauczył się języka C# w jego początkowej fazie rozwoju,
gdy pracował w siedzibie Microsoft w Redmond, USA, w latach 2001-2003.
W tym czasie był członkiem zespołu odpowiedzialnego za tworzenie
pierwszych oficjalnych kursów szkoleniowych dla języka C#, jeszcze
zanim został on oficjalnie wydany. Dzięki temu miał bezpośredni dostęp
```

```
do zespołów projektowych Microsoftu, co pozwoliło mu zdobyć dogłębną
wiedzę o języku I platformie .NET od samego początku ich istnienia.
```

12. Naciśnij klawisz *X*, aby zakończyć aplikację.

Spróbujmy teraz włączyć automatyczne wykorzystywanie wielu funkcji i dodajmy do kernela pamięć konwersacji, aby umożliwić naszemu chatbotowi korzystanie z poprzednich pytań i odpowiedzi w trakcie prowadzonej konwersacji.

Dodanie pamięci sesji i włączanie wielu funkcji

Obecnie nasz chatbot nie pamięta wcześniejszych pytań, co oznacza, że nie możemy prowadzić z nim interaktywnej rozmowy. Wyobraź sobie następującą interakcję:

```
Wpisz swoje pytanie: Opowiedz dowcip o programistach.
Jak programista zmienia żarówkę? Wcale, to problem sprzętowy!

Wprowadź swoje pytanie: Dlaczego to jest zabawne?
Umiejętność Marka dostrzegania zabawnych stron rzeczy dodaje uroku jego
profilowi.
```

Przy braku kontekstu następujące po sobie pytania mogą powodować generowanie dziwnych odpowiedzi przez model LLM. Zobaczmy, jak można zapewnić chatbotowi pamięć kontekstu:

1. Na początku pliku *Program.cs* zaimportuj przestrzeń nazw umożliwiającą pracę z historią chatu, tak jak w poniższym kodzie:

   ```
   // ChatHistory i podobne funkcje.
   using Microsoft.SemanticKernel.ChatCompletion;
   ```

2. W pliku *Program.cs* przed pętlą `while` dodaj instrukcje zapamiętywania poprzednich wiadomości z chatu i tak skonfiguruj model OpenAI, aby według potrzeby automatycznie wywoływał dowolne zarejestrowane funkcje. Następnie w pętli `while` umieść w komentarzu instrukcję wywołującą pojedynczą funkcję i dodaj instrukcje zapisujące wiadomości użytkownika i odpowiedzi modelu, zgodnie z wyróżnieniami w poniższym kodzie:

   ```
   IChatCompletionService completion =
     kernel.GetRequiredService<IChatCompletionService>();

   ChatHistory history = new(systemMessage: "Jesteś asystentem AI z wiedzą,
   ↪umiejętnościami i doświadczeniem Marka J. Price'a.");

   OpenAIPromptExecutionSettings options = new()
     { ToolCallBehavior = ToolCallBehavior.AutoInvokeKernelFunctions };

   while (key is not ConsoleKey.X)
   {
     Write("Wpisz swoje pytanie: ");
     string question = ReadLine()!;

     // WriteLine(await kernel.InvokePromptAsync(question));
   ```

```
    //arguments["question"] = question;
    //  Wywołaj jedną funkcję.
    //  WriteLine(await function.InvokeAsync(kernel, arguments));

    history.AddUserMessage(question);
    ChatMessageContent answer = await
      completion.GetChatMessageContentAsync(history);
    history.AddAssistantMessage(answer.Content!);
    WriteLine(answer.Content);

    WriteLine();
    WriteLine("Naciśnij klawisz X, aby zakończyć, albo inny klawisz,
    ↪żeby zadać kolejne pytanie.");
    key = ReadKey(intercept: true).Key;
}
```

3. Uruchom projekt *ChatApp* bez debugowania.
4. Wprowadź taką instrukcję: `Opowiedz dowcip o programistach`. Następnie przyjrzyj się odpowiedzi, która powinna być podobna do poniższej:

```
Dlaczego programiści mylą Halloween z Bożym Narodzeniem?
Bo 31 OCT == 25 DEC!
```

5. Wprowadź teraz pytanie: `Dlaczego to jest zabawne`? Ponownie przyjrzyj się odpowiedzi, podobnej do poniższej:

```
W systemie szesnastkowym (hexadecimal), liczba 31 (hex) w systemie
dziesiętnym to 49. Ale w systemie ósemkowym (octal), 31 (octal) odpowiada
liczbie 25 w dziesiętnym. Więc w pewnym sensie 31 OCT równa się 25 DEC,
jeśli patrzysz z perspektywy systemów liczbowych.
Dowcip jest grą na słowach i logice, którą docenią osoby techniczne,
znające systemy numeryczne.
```

6. Wprowadź kolejne pytanie: `Jakie owoce morza są najdroższe`? Przyjrzyj się odpowiedzi, takiej jak pokazana na poniższym wydruku:

```
Aby znaleźć najdroższy owoc morza, mogę pobrać informacje o produktach
z kategorii owoców morza dostępnych w bazie danych, a następnie
zidentyfikować produkt z najwyższą ceną. Pozwól, że zdobędę dla Ciebie
potrzebne dane. Najdroższym produktem z owoców morza są „Carnarvon Tigers"
z ceną jednostkową 62,5 $. Jest to najwyżej wyceniony produkt w bazie
danych. Jeśli masz więcej pytań lub potrzebujesz dalszych informacji,
daj znać!
```

7. Wprowadź następujące pytanie: `Jakie produkty znajdują się w kategorii owoców morza`? Ponownie przyjrzyj się odpowiedzi, która powinna wyglądać tak jak poniższa:

```
Oto produkty w kategorii owoce morza:
1. Ikura
2. Konbu
3. Carnarvon Tigers
4. Nord-Ost Matjeshering
5. Inlagd Sill
6. Gravad lax
```

```
7. Boston Crab Meat
8. Jack's New England Clam Chowder
9. Rogede sild
10. Spegesild
11. Escargots de Bourgogne
12. Röd Kaviar
```

8. Wprowadź jeszcze jedno pytanie: `Proszę, podaj też ceny`. Tym razem odpowiedź powinna zostać uzupełniona, tak jak na poniższym wydruku:

```
Oto produkty w kategorii owoce morza wraz z ich cenami:
1. Ikura - $31
2. Konbu - $6
3. Carnarvon Tigers - $62.5
4. Nord-Ost Matjeshering - $25.89
5. Inlagd Sill - $19
6. Gravad lax - $26
7. Boston Crab Meat - $18.4
8. Jack's New England Clam Chowder - $9.65
9. Rogede sild - $9.5
10. Spegesild - $12
11. Escargots de Bourgogne - $13.25
12. Röd Kaviar - $15
```

9. Naciśnij klawisz *X*, aby zakończyć aplikację.

Czasami trzeba chwilę poczekać na otrzymanie odpowiedzi z usługi OpenAI. Możemy jednak włączyć asynchroniczne przesyłanie wyników, by zapewnić użytkownikowi lepsze doświadczenia.

Strumieniowanie wyników

Zaimplementujmy teraz strumieniowanie wyników:

1. W pliku *Program.cs* przed pętlą `while` dodaj instrukcję definiującą obiekt `StringBuilder`, a następnie w pętli `while` dodaj instrukcje implementujące asynchroniczne strumieniowanie wyników, zgodnie z wyróżnieniami w poniższym kodzie:

```
// Podstawa implementacji asynchronicznego strumieniowania odpowiedzi.
StringBuilder builder = new();

while (key is not ConsoleKey.X)
{
  Write("Wpisz swoje pytanie: ");
  string question = ReadLine()!;

  // WriteLine(await kernel.InvokePromptAsync(question));
  //arguments["question"] = question;
  // Wywołaj jedną funkcję.
  //WriteLine(await function.InvokeAsync(kernel, arguments));

  history.AddUserMessage(question);
```

```
    // ChatMessageContent answer = await
    // completion.GetChatMessageContentAsync(history);

    builder.Clear();
    await foreach (StreamingChatMessageContent message
    in completion.GetStreamingChatMessageContentsAsync(history))
    {
      Write(message.Content);
      builder.Append(message.Content);
    }
    history.AddAssistantMessage(builder.ToString());

    WriteLine();
    WriteLine("Naciśnij klawisz X, aby zakończyć, albo inny klawisz, żeby
zadać kolejne pytanie.");
    key = ReadKey(intercept: true).Key;
}
```

2. Uruchom bez debugowania projekt *ChatApp*.
3. Zwróć uwagę na to, że odpowiedzi z usługi OpenAI są teraz strumieniowane asynchronicznie.

Dodawanie protokołowania i zwiększanie niezawodności

Teraz poprawmy chatbota, dodając możliwość protokołowania i zwiększając jego niezawodność:

1. W pliku projektu *ChatApp.csproj* dodaj odwołania do pakietów pozwalających na włączenie protokołowania, tak jak w poniższym kodzie:

```
<!-- Dodaj protokołowanie w konsoli i odporność HTTP. -->
<PackageReference Version="8.0.0" Include="Microsoft.Extensions.Http" />
<PackageReference Version="8.2.0"
Include="Microsoft.Extensions.Http.Resilience" />
<PackageReference Version="8.0.0"
Include="Microsoft.Extensions.Logging.Console" />
```

2. Na początku pliku *Program.GetKernel.cs* zaimportuj przestrzenie nazw związane z protokołowaniem, tak jak w poniższym kodzie:

```
using Microsoft.Extensions.DependencyInjection; // AddLogging.
using Microsoft.Extensions.Logging; // LogLevel.
```

3. W pliku *Program.GetKernel.cs* przed instrukcją tworzącą obiekt `kernel` dopisz instrukcje włączające protokołowanie i funkcje odporności, tak jak w poniższym kodzie:

```
// Dodaj protokołowanie i odporność.
kernelBuilder.Services.AddLogging(c =>
    c.AddConsole().SetMinimumLevel(LogLevel.Trace));
```

```
kernelBuilder.Services.ConfigureHttpClientDefaults(c =>
    c.AddStandardResilienceHandler());

Kernel kernel = kernelBuilder.Build();
```

4. Uruchom bez debugowania projekt *ChatApp* i zauważ, że od razu wyświetlane są dodatkowe informacje, tak jak na poniższym wydruku:

```
trce: Program[0]
      Created KernelFunction 'GetAuthorBiography' for 'GetAuthorBiography'
trce: Microsoft.SemanticKernel.KernelPromptTemplate[0]
      Extracting blocks from template: Author biography:
      ↪{{ authorInformation.getAuthorBiography }}.
      {{ $question }}
trce: Program[0]
      Created KernelFunction 'GetProductsInCategory'
      ↪for 'GetProductsInCategory'
Wpisz swoje pytanie:
```

5. Wprowadź następujące zapytanie: `Proszę, wygeneruj kod źródłowy HTML dla tabeli produktów z kategorii condiments. Uwzględnij przy tym cenę i liczbę sztuk w magazynie.`

6. Przyjrzyj się odpowiedzi, która powinna być podobna do tej z poniższego wydruku:

````
Pobrałem produkty w kategorii Przyprawy wraz z ich cenami i informacjami
o stanie magazynowym. Pozwól, że wygeneruję kod źródłowy HTML dla tabeli
wyświetlającej te informacje.
Oto kod źródłowy HTML dla tabeli wyświetlającej produkty w kategorii
Przyprawy wraz z ich cenami i liczbą sztuk w magazynie:

```html
<table>
  <thead>
    <tr>
      <th>Nazwa produktu</th>
      <th>Cena jednostkowa</th>
      <th>Sztuki w magazynie</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Syrop anyżowy</td>
      <td>$10.00</td>
      <td>13</td>
    </tr>
...
    <tr>
      <td>Oryginalna Frankfurter grüne Soße</td>
      <td>$13.00</td>
      <td>32</td>
    </tr>
  </tbody>
````
````

```
</table>
```
Ta tabela zawiera nazwę produktu, cenę jednostkową oraz liczbę sztuk
w magazynie dla każdego produktu w kategorii Condiments. Możesz dowolnie
modyfikować kod, jeśli zajdzie taka potrzeba.
```

7. Naciśnij klawisz *X*, aby zakończyć aplikację.

Teraz zobaczmy, jak użyć lokalnego modelu LLM zamiast modeli działających w chmurze.

# Korzystanie z lokalnych modeli LLM

W tym podrozdziale sprawdzimy, jak pobierać i uruchamiać wielkie modele językowe (LLM) na lokalnym komputerze.

## Hugging Face

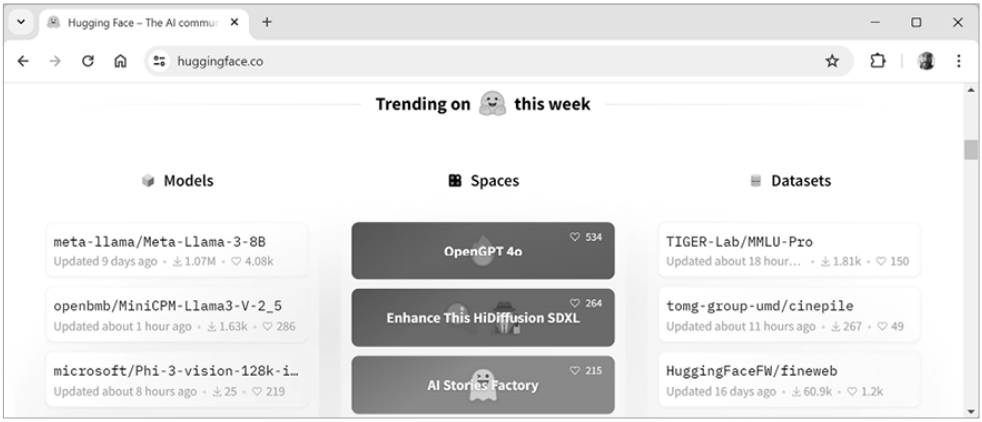
Hugging Face to popularna platforma open source, która umożliwia tworzenie i udostępnianie nowoczesnych modeli w dziedzinie przetwarzania języka naturalnego (NLP). Jest znana z opracowywania narzędzi, bibliotek i zasobów, które demokratyzują dostęp do zaawansowanych modeli uczenia maszynowego. Projekt Hugging Face w kilku kluczowych obszarach przyczynia się do rozwoju społeczności LLM:

- **Biblioteka Transformers.** Oferuje szeroki zakres wstępnie wytrenowanych modeli do różnych zadań NLP, takich jak generowanie tekstu, tłumaczenie, streszczanie i odpowiadanie na pytania.
- **Biblioteka Datasets.** Zapewnia ogromny zbiór gotowych do użycia zestawów danych do projektów uczenia maszynowego, obejmujących szeroki zakres zadań i dziedzin.
- **Model Hub.** To platforma, na której użytkownicy mogą przesyłać i udostępniać własne modele społeczności, przeglądać i pobierać modele udostępnione przez innych oraz korzystać z narzędzi do oceny i testowania wydajności modeli.
- **Biblioteka Tokenizers.** Udostępnia wydajne i elastyczne metody tokenizacji, które są wykorzystywane do przygotowywania danych tekstowych dla modeli bazujących na transformatorach.

**Uwaga**

Jedną z nowych funkcji, które zostaną wprowadzone w .NET 9 w listopadzie 2024 r., jest standardowy typ przeznaczony do pracy w bibliotekach tokenizacji. Więcej informacji na ten temat znajdziesz pod tym adresem: *https://github.com/dotnet/core/blob/main/release-notes/9.0/preview/preview4/libraries.md#tokenizer-library-enhancements*.
```

Możesz sprawdzić, co jest obecnie popularne na stronie głównej Hugging Face, używając adresu *https://huggingface.co/*. Widać tam, że model Llama 3 firmy Meta z 8 mld parametrów został pobrany ponad milion razy, co widać też na rysunku 9.7.



Rysunek 9.7. Strona główna Hugging Face

Uwaga

Model Llama 3 firmy Meta jest popularny z kilku powodów. Rozmiar 8 mld parametrów stanowi zrównoważoną wielkość modelu, co pozwala mu uchwycić subtelności w danych, na których został wytrenowany. Meta prawdopodobnie zastosowała zaawansowane metody szkoleniowe i zróżnicowane zestawy danych, aby zapewnić dobrą wydajność modelu w szerokim zakresie zadań. Dotychczasowa popularność powoduje dalsze zwiększanie popularności, ponieważ zachęca to do tworzenia silnego systemu wsparcia, w tym dokumentacji, forów i pomocy społeczności.

Teraz przyjrzymy się narzędziom, które mogą maksymalnie ułatwić pobieranie i używanie modeli Hugging Face na lokalnym komputerze.

Ollama

Ollama to program narzędziowy, którego zadaniem jest zarządzanie modelami LLM na urządzeniach lokalnych. Pozwala użytkownikom uruchamiać zaawansowane modele językowe, takie jak **LLaMA** (ang. *Large Language Model Meta AI*), bez konieczności polegania na rozwiązaniach działających w chmurze.

Dzięki lokalnemu uruchamianiu modeli Ollama sprawia, że wrażliwe dane pozostają na urządzeniu użytkownika, co zwiększa prywatność i bezpieczeństwo w porównaniu z usługami chmurowymi. Użytkownicy zachowują pełną własność i kontrolę nad swoimi danymi, co jest szczególnie ważne w przypadku informacji wrażliwych lub chronionych.

Deweloperzy i badacze mogą korzystać z Ollama do eksperymentowania i dostosowywania modeli LLM bez konieczności odwoływania się do zasobów chmurowych. Na dłuższą

metę uruchamianie modeli lokalnie może być bardziej opłacalne, ponieważ eliminuje potrzebę ciągłego subskrybowania usług. Z drugiej strony, instalacje lokalne potrzebują ciągłej konserwacji i aktualizacji, co wymaga dodatkowych nakładów w porównaniu do zarządzanych usług chmurowych.

Aby uruchomić LLM lokalnie, użytkownicy potrzebują wydajnego sprzętu, zwykle obejmującego zaawansowane procesory (CPU), znaczną ilość pamięci RAM oraz **karty graficzne** (GPU) do obsługi intensywnych obliczeń. Lokalne uruchamianie modeli zmniejsza jednak opóźnienia w generowaniu odpowiedzi, co jest istotne w aplikacjach wymagających natychmiastowych reakcji. Modele LLM są bardzo zasobożerne, a to oznacza, że nie każdy będzie dysponował sprzętem odpowiednim do ich efektywnego uruchamiania.

Projekt Ollama ma na celu uproszczenie procesu instalacji i konfiguracji modelu, udostępniając narzędzia i dokumentację, które pomagają użytkownikom szybko rozpocząć pracę.

Spróbujmy teraz pobrać i zainstalować pakiet Ollama:

1. Otwórz swoją przeglądarkę i przejdź na stronę *https://ollama.com/*.
2. Na stronie głównej kliknij przycisk *Download*.
3. Wybierz swój system operacyjny i postępuj zgodnie z instrukcjami instalacji.

Modele Ollama

Modele zwykle występują w rozmaitych wariantach, różniących się od siebie rozmiarem i możliwościami. Na przykład:

- `llama` i `llama:70b` różnią się rozmiarem.

Uwaga

Mniejsze modele, takie jak *llama*, mogą być wystarczające do wykonywania mniej złożonych zadań, jak podstawowe generowanie tekstu, proste odpowiadanie na pytania i podsumowywanie. W przypadku aplikacji wymagających odpowiedzi w czasie rzeczywistym, takich jak chatboty, mniejsze modele są zazwyczaj lepsze ze względu na mniejsze opóźnienia w procesie wnioskowania. Bardziej złożone zadania, takie jak dogłębne rozumienie, szczegółowe rozumowanie i generowanie dłuższych, spójnych tekstów, mogą wymagać użycia większych modeli, takich jak *llama:70b*. Jednak większe modele wymagają znacznie większej mocy obliczeniowej i pamięci zarówno podczas treningu, jak i przy wykonywaniu zadań. Warto również pamiętać, że w wybranych dziedzinach mniejszy, ale dobrze dostrojony model może przewyższyć większy, ogólny model.

- `mistral:instruct` — model wykonujący przekazane mu polecenia.
- `mistral:text` — model bazowy bez wykonanego dostrajania, najlepiej nadający się do prostego uzupełniania tekstu.

Aby szybko pobrać i uruchomić model Ollama w trybie interaktywnym, użyj następującego polecenia:

```
ollama run <model>
```

Najczęściej używane modele obsługiwane przez Ollama zostały zebrane w tabeli 9.1.

Tabela 9.1. Popularne modele Ollama

Model	Liczba parametrów	Wielkość	Opis
`llama:70b`	70 mld	40 GB	Najbardziej zaawansowany otwarty LLM dostępny do tej pory. Modele Llama 3 zostały dostrojone do prowadzenia dialogu i w tym zastosowaniu sprawdzają się lepiej niż jakiekolwiek inne otwarte modele
`llama3`	8 mld	4,7 GB	Mniejsza wersja modelu z rodziny Llama 3
`codellama`	7 mld	3,8 GB	Zajmuje się generowaniem i analizowaniem kodu na podstawie podpowiedzi tekstowych
`llama2-uncensored`	7 mld	3,8 GB	Nieocenzurowany model Llama 2, przygotowany przez George'a Sunga i Jarrada Hope'a. Wykorzystali oni proces zdefiniowany przez Erica Hartforda w tym artykule: *https://erichartford.com/uncensored-models*
`phi3`	3,8 mld	2,3 GB	Phi-3 to rodzina lekkich, małych (3 mld parametrów) i średnich (14 mld parametrów) modeli przygotowana przez Microsoft. Na konferencji Microsoft Build 2024 zaprezentowano wersję `phi3-silica`, która została zoptymalizowana do pracy z procesorami NPU (Neural Processing Unit) o wydajności większej niż 40 TOPS
`mistral`	7 mld	4,1 GB	Model o otwartej licencji Apache, dostępny w wersji wykonującej podane instrukcje i w wersji przygotowanej do uzupełniania tekstu
`gemma:7b`	7 mld	4,8 GB	Gemma to rodzina lekkich, nowoczesnych modeli przygotowanych przez Google DeepMind
`gemma:2b`	2 mld	1,4 GB	Najmniejszy model z rodziny Gemma odpowiedni dla urządzeń mobilnych

Uwaga

„Nieocenzurowany" w kontekście LLM oznacza, że model działa bez z góry narzuconych ograniczeń dotyczących generowanych treści, co pozwala na szerszy zakres wyników, ale zwiększa również ryzyko generowania szkodliwych lub nieodpowiednich treści. Z tego powodu modele nieocenzurowane mają większą wartość dla zadań związanych z badaniami i rozwojem, mogą pomagać komikom przygotowywać nowe dowcipy, ale też wymagają dokładniejszej kontroli, ze względu na problemy etyczne, jakie mogą się pojawiać w praktycznych zastosowaniach.

W naszym przypadku skorzystamy z modelu Llama 3, który jest wystarczająco wszechstronny i ma całkiem dużą szansę poprawnego działania na Twoim komputerze. Szczegóły licencji tego modelu znajdziesz pod adresem *https://llama.meta.com/llama3/license/*.

Interfejs wiersza poleceń Ollama

Interfejs wiersza poleceń Ollama udostępnia wiele różnych poleceń do lokalnego uruchamiania modeli LLM i zarządzania nimi. Co prawda poszczególne polecenia i opcje mogą się różnić w zależności od wersji i implementacji Ollama, ale poniżej przedstawiam ogólny przegląd najczęściej używanych poleceń.

Przyjrzyjmy się teraz możliwościom poleceń Ollama w połączeniu z modelem Llama 3:

1. W wierszu poleceń lub terminalu wpisz poniższe polecenie, aby sprawdzić wersję narzędzia:

```
ollama --version
```

2. Przyjrzyj się odpowiedzi, która powinna wyglądać mniej więcej tak:

```
ollama version is 0.1.38
```

3. Aby pobrać określony model, taki jak Llama 3, wpisz polecenie:

```
ollama pull llama3
```

4. Przyjrzyj się odpowiedzi, która powinna wyglądać tak:

```
pulling manifest
pulling 6a0746a1ec1a... 100% ▕▏ 4.7 GB
pulling 4fa551d4f938... 100% ▕▏  12 KB
pulling 8ab4849b038c... 100% ▕▏  254 B
pulling 577073ffcc6c... 100% ▕▏  110 B
pulling 3f8eb4da87fa... 100% ▕▏  485 B
verifying sha256 digest
writing manifest
removing any unused layers
success
```

Uwaga

Możesz też usunąć ten model za pomocą polecenia `ollama rm llama3`.

5. Aby wyświetlić listę dostępnych lokalnych modeli, wpisz polecenie:

```
ollama list
```

6. Przyjrzyj się odpowiedzi, która powinna wyglądać tak:

```
NAME          ID            SIZE      MODIFIED
llama3:latest 365c0bd3c000  4.7 GB    21 minutes ago
```

7. Aby uruchomić określony model (lub pobrać go, jeśli jeszcze nie został pobrany), wpisz poniższe polecenie w wierszu poleceń lub w terminalu:

```
ollama run llama3
```

8. W efekcie powinien się pojawić nowy wiersz poleceń, taki jak poniżej:

```
>>> Send a message (/? for help)
```

9. Wprowadź swoje pytanie, np. *Czym jest .NET?*, i przyjrzyj się odpowiedzi.
10. Aby zakończyć, wpisz polecenie /**bye**.

Uwaga

Wiersz poleceń Ollama oferuje również polecenia do kopiowania i tworzenia nowych modeli. Szczegółową dokumentację znajdziesz pod adresem *https://github.com/ollama/ollama#cli-reference*.

Projekt Ollama udostępnia biblioteki klienckie wyłącznie dla Pythona i JavaScriptu, ale programiści .NET przygotowali już dedykowane pakiety, którym przyjrzymy się w kolejnym punkcie.

Pakiet OllamaSharp dla .NET

OllamaSharp to projekt udostępniania API Ollama na platformie .NET, umożliwiający łatwą interakcję aplikacji .NET z modelami Ollama. Obsługuje wszystkie punkty końcowe API Ollama, w tym dostęp do chatów, osadzania, wyświetlania dostępnych modeli, pobierania i tworzenia nowych modeli.

Więcej informacji

Dodatkowe informacje znajdziesz w repozytorium OllamaSharp na GitHubie: *https://github.com/awaescher/OllamaSharp*.

Sprawdźmy, jak to działa:

1. W swoim edytorze kodu utwórz nowy projekt aplikacji konsoli o nazwie *OllamaApp* i dodaj go do rozwiązania *Rozdzial09*.
2. W pliku projektu *OllamaApp.csproj* dodaj odwołania do pakietów *Spectre Console* i *Ollama*, a także statycznie i globalnie zaimportuj klasę `System.Console`, zgodnie z poniższym kodem:

```
<ItemGroup>
  <PackageReference Include="Spectre.Console" Version="0.49.1" />
  <PackageReference Include="OllamaSharp" Version="1.1.9" />
</ItemGroup>

<ItemGroup>
  <Using Include="System.Console" Static="true" />
</ItemGroup>
```

Uwaga

Numery najnowszych wersji pakietów możesz sprawdzić pod adresami *https://www.nuget.org/packages/Spectre.Console* i *https://www.nuget.org/packages/OllamaSharp*.

3. W pliku *Program.cs* usuń istniejące instrukcje i dopisz kod:
   - Importujący przestrzenie nazw potrzebne do pracy z Ollama i Spectre Console.
   - Tworzący klienta Ollama wysyłającego żądania do lokalnego hosta na domyślnym porcie 11434.
   - Wyświetlający tabelę Spectre Console z modelami dostępnymi w lokalnej instalacji Ollama.
   - Nakazujący klientowi Ollama używanie najnowszego modelu Llama 3.
   - Pobierający zapytanie od użytkownika, wysyłający je do klienta Ollama i wyświetlający odpowiedź.
   - Mierzący czas pracy modelu za pomocą stopera.

```
using OllamaSharp; // OllamaApiClient.
using OllamaSharp.Models; // Model.
using Spectre.Console; // Table.
using System.Diagnostics; // Stopwatch.

// Domyślny port API Ollama to 11434.
string port = "11434";
Uri uri = new($"http://localhost:{port}");

OllamaApiClient ollama = new(uri);

Table table = new();
table.AddColumn("Nazwa");
table.AddColumn("Wielkość");

// Pobierz listę modeli.
IEnumerable<Model> models = await ollama.ListLocalModels();

foreach (Model model in models)
{
  table.AddRow(model.Name, model.Size.ToString("N0"));
}

AnsiConsole.Write(table);

string modelName = "llama3:latest";

WriteLine();
WriteLine($"Wybrany model: {modelName}");
ollama.SelectedModel = modelName;
```

```
Write("Wpisz swój prompt: ");
string? prompt = ReadLine();
if (string.IsNullOrWhiteSpace(prompt))
{
  WriteLine("Musisz podać swój prompt. Zamykam aplikację.");
  return;
}

Stopwatch timer = Stopwatch.StartNew();

ConversationContext context = new([]);
context = await ollama.StreamCompletion(
  prompt, context, stream => Write(stream.Response));

timer.Stop();

WriteLine();
WriteLine();
WriteLine($"Czas wykonania: {timer.ElapsedMilliseconds:N0} ms");
```

4. Uruchom bez debugowania projekt *OllamaApp*.
5. Wprowadź zapytanie, np. `Czym jest .NET?`, i przejrzyj otrzymaną odpowiedź:

```
Wpisz swój prompt: Czym jest .NET?
.NET (pierwotnie znany jako Common Language Runtime, CLR) to framework
programistyczny stworzony przez firmę Microsoft. Jest to platforma,
która umożliwia tworzenie aplikacji komputerowych w różnych językach
programowania, takich jak C#, F#, Visual Basic .NET, itp.
...
Czas wykonania: 170 501 ms
```

Przyjrzyjmy się teraz innym narzędziom do eksperymentowania z lokalnymi LLM.

LM Studio

Używając LM Studio, możesz:

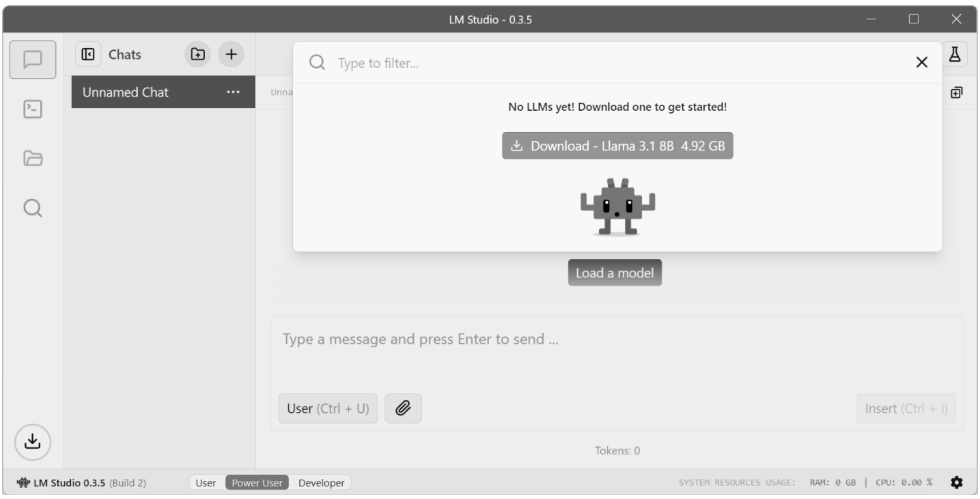
- Uruchamiać modele LLM na swoim laptopie bez dostępu do internetu.
- Korzystać z tych modeli za pomocą interfejsu chatu w aplikacji lub lokalnego serwera zgodnego z OpenAI.
- Pobierać kompatybilne pliki modeli z repozytoriów Hugging Face.
- Odkrywać nowe i interesujące modele LLM na stronie głównej aplikacji.

Uwaga

Minimalne wymagania to komputer Mac z procesorem M1 lub komputer z systemem Windows i procesorem obsługującym zestaw instrukcji AVX2. Wersja dla systemu Linux jest obecnie w fazie beta.

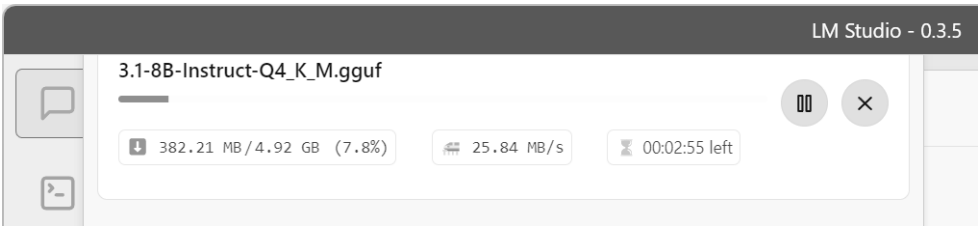
Aby zacząć używać LM Studio:

1. W przeglądarce otwórz stronę *https://lmstudio.ai/*.
2. Aby pobrać LM Studio, kliknij przycisk właściwy dla Twojego systemu operacyjnego.
3. Uruchom pobraną aplikację.
4. Na stronie głównej aplikacji zobaczysz model Llama 3 o 8 mld parametrów (**8B**) przygotowany przez firmę Meta AI, ponieważ jest to jeden z najpopularniejszych modeli. (Jeżeli aktualnie nie jest najpopularniejszy, skorzystaj z funkcji wyszukiwania, aby znaleźć ten lub inny model). Kliknij przycisk *Download*, jak na rysunku 9.8.



Rysunek 9.8. Strona główna LM Studio z modelem Llama 3 8B firmy Meta AI

5. Poczekaj na pobranie modelu. Postęp pobierania można monitorować, rozwijając obszar statusu, jak na rysunku 9.9.



Rysunek 9.9. Pobieranie modelu Llama 3 8B w LM Studio

6. Przejdź do sekcji *AI Chat* i wybierz pobrany model.
7. W polu *USER* wpisz zapytanie, np. **`Czym jest .NET?`**, i przyjrzyj się odpowiedzi, która powinna być dość wyczerpująca.

8. W polu *USER* wpisz inne zapytanie, np. **`Kto mówi "Dlaczego Gamora?" i dlaczego?`**
9. Zwróć uwagę, że odpowiedź może zawierać halucynacje, jak w poniższym przykładzie:

```
To pytanie odnosi się do popularnego filmu Marvela, "Guardians of the
Galaxy". W jednej ze scen, główny bohater Peter Quill (Star-Lord)
otrzymuje zlecenie zniszczenia planety Xandar i pyta "Dlaczego Gamora?",
co oznacza, że chce wiedzieć dlaczego Gamora, strzelająca do niego
z pistoletu. Głębsze znaczenie tej kwestii polega na tym, że Peter Quill,
jako syn Zarekana, jest pozbawiony swojej tożsamości i rodziny, a Gamora,
która go strzeliła w dzieciństwie, jest dla niego symbolem cierpienia
I utraconego początku.
```

10. Jeżeli zadasz to samo pytanie ChatGPT-4, otrzymasz znacznie lepszą odpowiedź:

```
Cytat "Dlaczego Gamora?" pochodzi z filmu "Avengers: Wojna bez granic"
(Avengers: Infinity War) i wypowiada go Drax, grany przez Dave'a Bautistę.
Scena, w której pada ten tekst, jest humorystycznym momentem podczas
konfrontacji między Strażnikami Galaktyki a Iron Manem, Doktorem
Strange'em i Spider-Manem.
```

Jeżeli porównasz lokalny model z modelem w chmurze, to zauważysz, że modele lokalne są znacznie wolniejsze i mniej precyzyjne, choć zależy to też od możliwości Twojego sprzętu. Jednak wraz z rozwojem możliwości sztucznej inteligencji, przy wykorzystaniu urządzeń takich jak **Copilot+ PC** (np. Surface Pro 11 i Surface Laptop 7 z chipami Snapdragon X Elite lub Plus), ta sytuacja powinna ulec poprawie. Planuję przetestować, jak szybko te nowe urządzenia działają z lokalnymi modelami LLM, ale mój nowy laptop dotrze do mnie dopiero 18 czerwca 2024 r., po zakończeniu ostatnich rozdziałów tej książki.

Więcej informacji

Dodatkowe informacje na temat urządzeń Copilot+ PC znajdziesz pod tym adresem: *https://blogs.microsoft.com/blog/2024/05/20/introducing-copilot-pcs/*.

Ćwiczenia i dalsza nauka

Sprawdź swoją wiedzę, odpowiadając na kilka pytań dotyczących tematów poruszonych w tym rozdziale. Przeprowadź praktyczne ćwiczenia i poszerz swoją wiedzę poprzez zgłębianie tych zagadnień.

Ćwiczenie 9.1. Materiały dostępne wyłącznie online

Wprowadzenie do Semantic Kernel: *https://devblogs.microsoft.com/semantic-kernel/hello-world/*.

Artykuł *Semantic Kernel Embeddings and Memories*: *https://devblogs.microsoft.com/semantic-kernel/semantic-kernel-embeddings-and-memories-explore-github-repos-with-chat-ui/*.

Artykuł na temat poprawiania jednoznaczności i inteligencji aplikacji używających AI za pomocą Redisa: *https://devblogs.microsoft.com/semantic-kernel/making-ai-powered-net-apps-more-consistent-and-intelligent-with-redis/*.

Wpis na blogu *Building AI-powered Microsoft Copilot with SignalR and other open-source tools*: *https://devblogs.microsoft.com/dotnet/building-ai-powered-bing-chat-with-signalr-and-other-open-source-tools/*.

Wpis na blogu *Build Intelligent Applications using ChatGPT & Azure Cosmos DB*: *https://devblogs.microsoft.com/cosmosdb/chatgpt-azure-cosmos-db/*.

Korzystanie z modeli Hugging Face z API Semantic Kernel to potężny sposób na budowanie precyzyjnych i wydajnych aplikacji przetwarzania języka naturalnego (NLP). Wykorzystując mocne strony obu narzędzi, programiści mogą tworzyć znacznie bardziej efektywne aplikacje NLP. Więcej na ten temat możesz przeczytać w tym artykule: *https://devblogs.microsoft.com/semantic-kernel/how-to-use-hugging-face-models-with-semantic-kernel/*.

Ćwiczenie 9.2. Ćwiczenia praktyczne

Oto kilka pomysłów na inne usługi wykorzystujące modele językowe, które możesz spróbować przygotować:

1. Asystent naukowy przeszkolony na wielu artykułach, udzielający odpowiedzi z dokładnymi cytatami.
2. Chatbot programistyczny, który czyta e-booki programistyczne w formacie PDF z folderu i umożliwia prowadzenie interaktywnej rozmowy na ich temat. Jeżeli napotkasz problem na konkretnej stronie, możesz powiedzieć chatbotowi, o który fragment chodzi, pokazać komunikat o błędzie i otrzymać sensowną sugestię, jak rozwiązać ten problem.
3. Chatbot pomagający programiście .NET przygotować się do rozmowy kwalifikacyjnej.
4. Chatbot projektowy do wprowadzania nowych członków zespołu, przeszkolony na podstawie specyfikacji projektu, dokumentacji i bazy kodu, który może zaprezentować projekt nowym programistom i odpowiadać na pytania dotyczące tego projektu. Dzięki temu lider projektu nie musi odpowiadać na takie pytania na Slacku.
5. Asystent twórcy artykułów na bloga, który sugeruje tematy wpisów na bloga z poradami dotyczącymi .NET. Wybierasz jeden temat, a on pomaga Ci go napisać i następnie publikuje go na platformach LinkedIn, X, Threads i Mastodon.

Ćwiczenie 9.3. Sprawdź swoją wiedzę

Odpowiedz na poniższe pytania. Jeśli napotkasz problemy, spróbuj wyszukać odpowiedzi w internecie. Jeśli nadal będziesz miał trudności, odpowiedzi znajdują się w dodatku.

1. Czym jest Semantic Kernel?
2. Co robi metoda `CreateFunctionFromMethod`?
3. Domyślnie każdy prompt wysyłany do modelu LLM za pomocą biblioteki Semantic Kernel jest niezależny od pozostałych promptów. Jak dodać pamięć sesji do chatu, aby model zapamiętywał wcześniejsze prompty?
4. Domyślnie musisz czekać, aż cała odpowiedź zostanie zwrócona przez model LLM. Jak włączyć strumieniowanie odpowiedzi?
5. Czym jest Hugging Face?

Ćwiczenie 9.4. Dalsza lektura

Użyj linków z poniższej strony, aby dowiedzieć się więcej na tematy poruszane w tym rozdziale: *https://github.com/markjprice/tools-skills-net8/blob/main/docs/book-links.md#chapter-9---building-a-custom-llm-based-chat-service*.

Podsumowanie

W tym rozdziale dowiedzieliśmy się:

- Jak działają modele LLM i jak uzyskać do nich dostęp.
- Jak używać biblioteki Semantic Kernel.
- Jak rozbudowywać modele LLM za pomocą funkcji.
- Jak dodawać pamięć sesji i strumieniować wyniki.
- Czym jest Hugging Face i jak używać modeli lokalnie.

W następnym rozdziale poznamy kontenery wstrzykiwania zależności (ang. *dependency injection* — DI), które automatyzują proces zarządzania zależnościami i czasem życia usług.

Skorowidz

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

Ś

T

U

V

W

X

Y

Z

Od juniora do eksperta. Wszystko, co musisz wiedzieć o .NET!

Większość książek programistycznych obejmuje jedynie wybrane tematy, takie jak bezpieczeństwo, testowanie aplikacji czy wdrażanie w chmurze. Inne koncentrują się na architekturze aplikacji, wzorcach projektowych albo przygotowaniu się do rozmów kwalifikacyjnych. Trudno jednak znaleźć przystępny przewodnik, który pozwoliłby krok po kroku przyswoić wszystkie umiejętności zawodowego programisty .NET.

Dzięki temu podręcznikowi odblokujesz swój potencjał i otworzysz drogę do kariery. Nauczysz się zarządzania kodem źródłowym przy użyciu Gita i skutecznego nawigowania po projektach. Odkryjesz zaawansowane techniki debugowania i dokumentowania kodu, co poprawi jego czytelność i ułatwi utrzymanie projektów. Zgłębisz też tajniki kryptografii, by zapewnić poufność i spójność danych na każdym etapie cyklu życia aplikacji. W książce nie zabrakło wnikliwego spojrzenia na nowoczesne zagadnienia, takie jak budowanie inteligentnych aplikacji korzystających z modeli LLM, wstrzykiwanie zależności, testowanie czy konteneryzacja za pomocą Dockera. Dzięki wskazówkom dotyczącym najlepszych praktyk architektury oprogramowania zaczniesz tworzyć solidne, skalowalne i łatwe w utrzymaniu aplikacje!

W książce:

- zaawansowane techniki debugowania
- ochrona danych i aplikacji przy użyciu kryptografii
- zastosowanie modeli LLM i programowanie chmurowe z .NET Aspire
- wstrzykiwanie zależności
- optymalizacja wydajności poprzez benchmarking i testowanie
- przygotowanie się do rozmowy kwalifikacyjnej

Mark J. Price specjalizuje się w programowaniu w języku C#. Pracuje w Microsofcie, tworzy rozwiązania dla Microsoft Azure. Zdał ponad 80 egzaminów Microsoftu. Zajmuje się też dydaktyką: przeszkolił wielu przyszłych programistów, od 16-letnich początkujących aż po 70-latków na emeryturze.

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 978-83-289-2233-4
9 788328 922334
Cena: 139,00 zł
packt