

IA para desenvolvedores do .NET

Saiba como usar a IA com o .NET. Procure código de exemplo, tutoriais, guias de início rápido, artigos conceituais, entre outros.

Introdução

COMEÇAR AGORA

[Desenvolver aplicativos .NET](#)

[Exemplos e recursos de aprendizagem](#)

[Criar um aplicativo de chat de IA do Azure com o .NET](#)

[Resumir texto usando um aplicativo de chat do OpenAI do Azure](#)

[Gerar imagens usando a IA do Azure com o .NET](#)

Conceitos básicos

CONCEITO

[Como a IA generativa e os LLMs funcionam](#)

[Noções básicas de tokens](#)

[Preservar o significado semântico com inserções](#)

[Pesquisa semântica com bancos de dados vetoriais](#)

[Engenharia de prompts](#)

Tarefas comuns

GUIA DE INSTRUÇÕES

[Autenticar o Serviço de Aplicativo no OpenAI do Azure](#)

[Autenticar o Serviço de Aplicativo em um banco de dados vetorial](#)

[Usar o Redis com o SDK do Kernel Semântico](#)

[Usar modelos de IA personalizados e locais com o SDK do Kernel Semântico](#)

Tutoriais



TUTORIAL

[Dimensionar o OpenAI do Azure com Aplicativos de Contêiner do Azure](#)

[Exemplo de chat corporativo do .NET usando o RAG](#)

[Implementar o RAG usando a busca em vetores](#)

Treinamento



TUTORIAL

[Conceitos básicos do Serviço OpenAI do Azure](#)

[Gerar conversas Conclusões do OpenAI do Azure](#)

[Exemplo de chat corporativo do .NET usando o RAG](#)

[Desenvolver agentes de IA usando o OpenAI do Azure](#)

Desenvolver aplicativos .NET com recursos de IA

Artigo • 30/07/2024

Com o .NET, você pode usar a IA (inteligência artificial) para automatizar e realizar tarefas complexas em seus aplicativos usando as ferramentas, plataformas e serviços que são familiares para você.

Por que escolher .NET para criar aplicativos de IA?

Milhões de desenvolvedores usam o .NET para criar aplicativos executados na Web, em dispositivos móveis e desktop ou na nuvem. Usando o .NET para integrar a IA em seus aplicativos, você pode aproveitar tudo o que o .NET tem a oferecer:

- Uma história unificada para a criação de interfaces do usuário da Web, APIs e aplicativos.
- Desenvolva x64 no Windows, no macOS e no Linux
- De software livre e voltado para a comunidade.
- É executado em cima dos servidores Web e plataformas de nuvem mais populares.
- Ferramentas avançadas para editar, depurar, testar e implantar.

O que você pode criar com IA e .NET?

As oportunidades com IA são quase infinitas. Aqui estão alguns exemplos de soluções que você pode criar usando IA em seus aplicativos .NET:

- Processamento de linguagem: crie agentes virtuais/chatbots para conversar com seus dados e gerar conteúdo e imagens.
- Pesquisa visual computacional: identifique objetos em um objeto ou vídeo.
- Geração de áudio: use vozes sintetizadas para interagir com os clientes.
- Classificação: rotule a gravidade de um problema relatado pelo cliente.
- Automação de tarefas: execute automaticamente a próxima etapa em um fluxo de trabalho à medida que as tarefas são concluídas.

Caminho de aprendizado recomendado

Recomendamos a seguinte sequência de tutoriais e artigos para obter uma introdução ao desenvolvimento de aplicativos com IA e .NET:

 Expandir a tabela

Cenário	Tutorial
Criar um aplicativo de chat	Criar um aplicativo de chat de IA do Azure com o .NET
Resumir textos	Resumir texto usando o aplicativo de chat de IA do Azure com o .NET
Converse com seus dados	Obter informações sobre os seus dados de um aplicativo de chat de IA do Azure do .NET
Chamar funções do .NET com IA	Estender a IA do Azure usando ferramentas e executar uma função local com o .NET
Gerar imagens	Gerar imagens usando a IA do Azure com o .NET
Treinar seu próprio modelo	Tutorial do ML.NET

Navegue pelo sumário para saber mais sobre os principais conceitos, começando com [Como a IA gerativa e os LLMs funcionam](#).

Próximas etapas

- Início Rápido – criar um aplicativo de chat de IA do Azure com o .NET
- Série de vídeos: Machine Learning e IA com o .NET

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).



Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 Abrir um problema de documentação

 Fornecer comentários sobre o produto

Visão geral do ecossistema .NET + AI

Artigo • 24/11/2024

O ecossistema do .NET fornece muitas ferramentas, bibliotecas e serviços avançados para desenvolver aplicativos de IA. O .NET dá suporte a conexões de modelo de IA locais e de nuvem, muitos SDKs diferentes para vários serviços de banco de dados vetorial e de IA e outras ferramentas para ajudar você a criar aplicativos inteligentes de escopo e complexidade diferentes.

ⓘ Importante

Nem todos os SDKs e serviços apresentados neste documento são mantidos pela Microsoft. Ao considerar um SDK, avalie a qualidade, licenciamento, suporte e compatibilidade dele para garantir que ele atenda às suas necessidades.

Microsoft.Extensions.AI biblioteca para .NET

[Microsoft.Extensions.AI](#) é um conjunto de bibliotecas principais do .NET criadas em colaboração com desenvolvedores em todo o ecossistema .NET, incluindo o Kernel Semântico. Essas bibliotecas fornecem uma camada unificada de abstrações C# para interagir com serviços de IA, como modelos de linguagem pequenos e grandes (SLMs e LLMs), inserções e middleware.

[Microsoft.Extensions.AI](#) fornece abstrações que podem ser implementadas por vários serviços, todos aderindo aos mesmos conceitos básicos. Esta biblioteca não se destina a fornecer APIs personalizadas para os serviços de nenhum provedor específico. O objetivo é [Microsoft.Extensions.AI](#) atuar como uma camada unificadora dentro do ecossistema .NET, permitindo que os desenvolvedores escolham suas estruturas e bibliotecas preferidas, garantindo integração e colaboração perfeitas em todo o ecossistema.

Kernel Semântico para .NET

O [Kernel Semântico](#) é um SDK de código aberto que permite recursos de orquestração e integração de IA em seus aplicativos .NET. Esse SDK geralmente é a ferramenta de orquestração de IA recomendada para aplicativos .NET que usam um ou mais serviços de IA em combinação com outras APIs ou serviços Web, armazenamentos de dados e código personalizado. O Kernel Semântico beneficia os desenvolvedores empresariais das seguintes maneiras:

- Simplifica a integração de recursos de IA em aplicativos existentes para permitir uma solução coesa para produtos empresariais.

- Minimiza a curva de aprendizado do trabalho com diferentes modelos ou serviços de IA, fornecendo abstrações que reduzem a complexidade.
- Aumenta a confiabilidade ao reduzir o comportamento imprevisível dos prompts e das respostas dos modelos de IA. É possível ajustar os prompts e planejar tarefas para criar uma experiência de usuário controlada e previsível.

Para obter mais informações, consulte a documentação do [Semantic Kernel](#).

SDKs do .NET para a criação de aplicativos de IA

Vários SDKs diferentes estão disponíveis para o .NET criar aplicativos com recursos de IA, dependendo da plataforma de destino ou do modelo de IA. Os modelos da OpenAI oferecem funcionalidades de IA generativa avançadas, enquanto outros Serviços de IA do Azure fornecem soluções inteligentes para uma variedade de cenários específicos.

SDKs do .NET para modelos da OpenAI

[+] [Expandir a tabela](#)

Pacote NuGet	Modelos com suporte	Mantenedor ou fornecedor	Documentação
Microsoft.SemanticKernel	Modelos de OpenAI Modelos com suporte do OpenAI do Azure	Kernel Semântico (Microsoft)	Documentação do Kernel Semântico
SDK do OpenAI do Azure	Modelos com suporte do OpenAI do Azure	SDK do Azure para .NET (Microsoft)	Documentação dos serviços de OpenAI do Azure
SDK de OpenAI	Modelos com suporte da OpenAI	SDK da OpenAI do .NET (OpenAI)	Documentação dos serviços OpenAI

SDKs do .NET para Serviços de IA do Azure

O Azure oferece muitos outros serviços de IA para criar recursos de aplicativo e fluxos de trabalho específicos. A maioria desses serviços fornece um SDK do .NET para integrar suas funcionalidades em aplicativos personalizados. Alguns dos serviços mais usados são mostrados na tabela a seguir. Para obter uma lista completa de serviços disponíveis e recursos de aprendizagem, consulte a documentação [Serviços de IA do Azure](#).

[+] [Expandir a tabela](#)

Serviço	Descrição
Azure AI Search	Insira a pesquisa em nuvem habilitada para IA nos seus aplicativos Web e móveis.
Segurança de Conteúdo da IA do Azure	Detectar conteúdo indesejado ou ofensivo.
IA do Azure para Informação de Documentos	Transforme documentos em soluções inteligentes baseadas em dados.
Linguagem de IA do Azure	Criar aplicativos com recursos líderes do setor de reconhecimento de linguagem natural.
Fala de IA do Azure	Conversão de fala em texto, conversão de texto em fala, tradução e reconhecimento de locutor.
Tradutor de IA do Azure	Tecnologia de tradução baseada em IA com suporte para mais de 100 idiomas e dialetos.
Visão de IA do Azure	Analisa o conteúdo em imagens e vídeos.

Desenvolver com modelos de IA locais

Os aplicativos .NET também podem se conectar a modelos de IA locais para muitos cenários de desenvolvimento diferentes. O [Kernel Semântico](#) é a ferramenta recomendada para se conectar a modelos locais usando o .NET. O Kernel Semântico pode se conectar a vários modelos diferentes hospedados em uma variedade de plataformas e abstrai os detalhes de implementação de nível inferior.

Por exemplo, você pode usar o [Ollama](#) para [se conectar a modelos de IA locais com .NET](#), incluindo vários SLMs (Small Language Models) desenvolvidos pela Microsoft:

 Expandir a tabela

Modelo	Descrição
Modelos phi3	Uma família de SLMs poderosos com desempenho inovador a baixo custo e baixa latência.
Modelos orca	Modelos de pesquisa em tarefas como raciocínio sobre dados determinados pelo usuário, compreensão de leitura, resolução de problemas matemáticos e resumo de texto.

⚠️ Observação

Os SLMs anteriores também podem ser hospedados em outros serviços, como o Azure.

Conectar-se a serviços e bancos de dados vetoriais

Os aplicativos de IA geralmente usam bancos de dados e serviços de vetor de dados para melhorar a relevância e fornecer funcionalidade personalizada. Muitos desses serviços fornecem um SDK nativo para .NET, enquanto outros oferecem um serviço REST ao qual você pode se conectar por meio de um código personalizado. O Kernel Semântico fornece um modelo de componente extensível que permite que você use repositórios de vetores diferentes sem a necessidade de aprender cada SDK.

O Kernel Semântico fornece conectores para os seguintes bancos de dados e serviços de vetor:

 Expandir a tabela

Serviço vetorial	Conector do Kernel Semântico	SDK .NET
Azure AI Search	Microsoft.SemanticKernel.Connectors.AzureAISeach	Azure.Search.Documents
Azure Cosmos DB para NoSQL	Microsoft.SemanticKernel.Connectors.AzureCosmosDBNoSQL	Microsoft.Azure.Cosmos
Azure Cosmos DB for MongoDB	Microsoft.SemanticKernel.Connectors.AzureCosmosDBMongoDB	MongoDb.Driver
Servidor PostgreSQL do Azure	Microsoft.SemanticKernel.Connectors.Postgres	Npgsql
Banco de Dados SQL do Azure	Microsoft.SemanticKernel.Connectors.SqlServer	Microsoft.Data.SqlClient
Chroma	Microsoft.SemanticKernel.Connectors.Chroma	NA
DuckDB	Microsoft.SemanticKernel.Connectors.DuckDB	DuckDB.NET.Data.Full
Milvus	Microsoft.SemanticKernel.Connectors.Milvus	Milvus.Client
Busca em vetores do MongoDB Atlas	Microsoft.SemanticKernel.Connectors.MongoDB	MongoDb.Driver

Serviço vetorial	Conector do Kernel Semântico	SDK .NET
Pinecone	Microsoft.SemanticKernel.Connectors.Pinecone ↗	REST API ↗
Postgres	Microsoft.SemanticKernel.Connectors.Postgres ↗	Npgsql ↗
Qdrant	Microsoft.SemanticKernel.Connectors.Qdrant ↗	Qdrant.Client ↗
Redis	Microsoft.SemanticKernel.Connectors.Redis ↗	StackExchange.Redis ↗
Weaviate	Microsoft.SemanticKernel.Connectors.Weaviate ↗	REST API ↗

Acesse a documentação de cada serviço respectivo para descobrir o SDK do .NET e o suporte à API.

Outras opções

Este artigo resumiu as ferramentas e os SDKs no ecossistema do .NET, com foco nos serviços que fornecem suporte oficial ao .NET. Dependendo de suas necessidades e estágio de desenvolvimento de aplicativos, talvez você também queira dar uma olhada nas opções de software livre para o ecossistema em [a lista não oficial de recursos .NET + IA](#) ↗. A Microsoft não é a mantenedora de muitos desses projetos, portanto, certifique-se de examinar sua qualidade, licenciamento e suporte.

Próximas etapas

- [o que é Kernel Semântico?](#)
- [Início rápido – resumir texto usando o aplicativo de chat de IA do Azure com o .NET](#)

Blocos de construção de IA unificados para .NET usando Microsoft.Extensions.AI

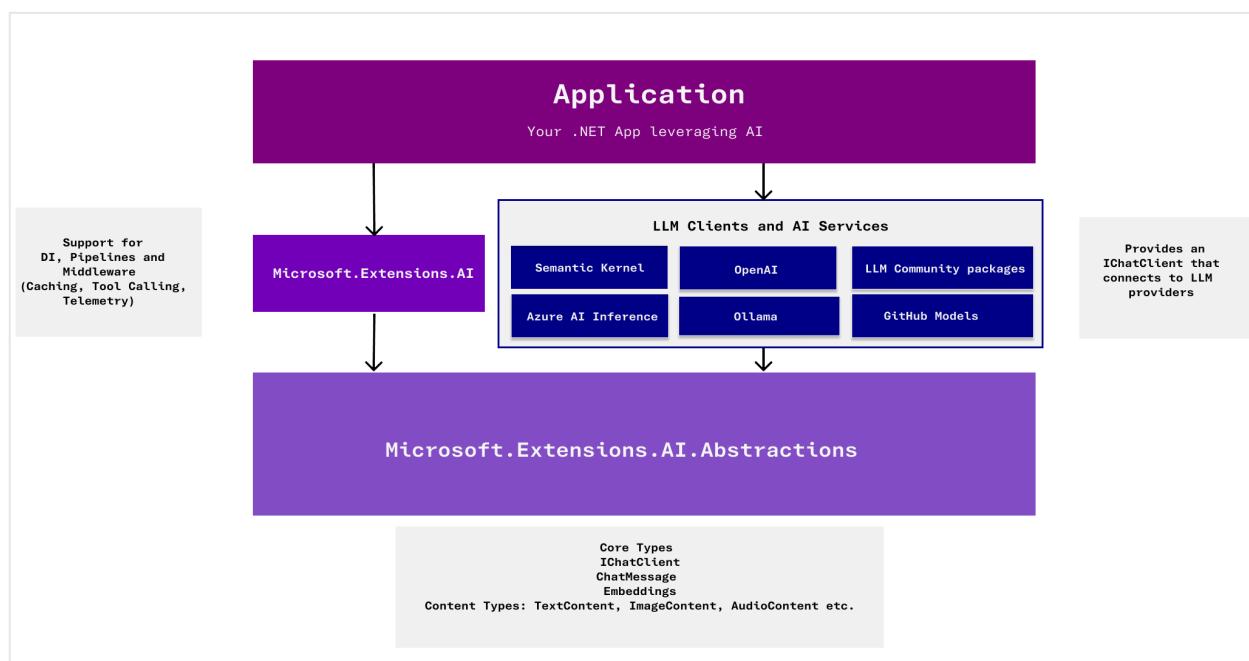
Artigo • 13/11/2024

O ecossistema .NET fornece abstrações para integrar serviços de IA em aplicativos e bibliotecas .NET usando as [Microsoft.Extensions.AI](#) bibliotecas and [Microsoft.Extensions.AI.Abstractions](#). A equipe do .NET também aprimorou as bibliotecas principais [Microsoft.Extensions.*](#) com essas abstrações para aplicativos e bibliotecas de IA generativa do .NET. Nas próximas seções, você aprenderá:

- Principais conceitos e recursos das [Microsoft.Extensions.AI](#) bibliotecas.
- Como trabalhar com abstrações de IA em seus aplicativos e os benefícios que eles oferecem.
- Conceitos essenciais de middleware de IA.

O que é a biblioteca Microsoft.Extensions.AI?

[Microsoft.Extensions.AI](#) é um conjunto de bibliotecas principais do .NET criadas em colaboração com desenvolvedores em todo o ecossistema .NET, incluindo o Kernel Semântico. Essas bibliotecas fornecem uma camada unificada de abstrações C# para interagir com serviços de IA, como modelos de linguagem pequenos e grandes (SLMs e LLMs), inserções e middleware.



`Microsoft.Extensions.AI` fornece abstrações que podem ser implementadas por vários serviços, todos aderindo aos mesmos conceitos básicos. Esta biblioteca não se destina a fornecer APIs personalizadas para os serviços de nenhum provedor específico. O objetivo é `Microsoft.Extensions.AI` atuar como uma camada unificadora dentro do ecossistema .NET, permitindo que os desenvolvedores escolham suas estruturas e bibliotecas preferidas, garantindo integração e colaboração perfeitas em todo o ecossistema.

Trabalhar com abstrações para serviços comuns de IA

Os recursos de IA estão evoluindo rapidamente, com padrões emergentes para funcionalidades comuns:

- Recursos de bate-papo para solicitar uma IA de conversação para análise de informações ou dados.
- Geração de incorporação para integração com recursos de pesquisa vetorial.
- Chamada de ferramenta para integração com outros serviços, plataformas ou código.

A `Microsoft.Extensions.AI` biblioteca fornece abstrações para esses tipos de tarefas, para que os desenvolvedores possam se concentrar na codificação em relação aos recursos conceituais de IA, em vez de plataformas específicas ou implementações de provedores. Abstrações unificadas são cruciais para que os desenvolvedores trabalhem efetivamente em diferentes fontes.

Por exemplo, a interface permite o `IChatClient` consumo de modelos de linguagem de vários provedores, esteja você se conectando a um serviço OpenAI do Azure ou executando uma instalação local do Ollama. Qualquer pacote .NET que forneça um cliente de IA pode implementar a interface, permitindo a integração perfeita com o `IChatClient` consumo de código .NET:

C#

```
IChatClient client =
    environment.IsDevelopment ?
        new OllamaChatClient(...) :
        new AzureAIIInferenceChatClient(...);
```

Em seguida, independentemente do provedor que você está usando, você pode enviar solicitações da seguinte maneira:

C#

```
var response = await chatClient.CompleteAsync(  
    "Translate the following text into Pig Latin: I love .NET and AI");  
  
Console.WriteLine(response.Message);
```

Essas abstrações permitem código C# idiomático para vários cenários com alterações mínimas de código, se você estiver usando serviços diferentes para desenvolvimento e produção, abordando cenários híbridos ou explorando outros provedores de serviços.

Os autores de bibliotecas que implementam essas abstrações tornam seus clientes interoperáveis com o ecossistema mais amplo `Microsoft.Extensions.AI`. As APIs específicas do serviço permanecem acessíveis, se necessário, permitindo que os consumidores codifiquem em relação às abstrações padrão e passem para APIs proprietárias somente quando necessário.

`Microsoft.Extensions.AI` Fornece implementações para os seguintes serviços por meio de pacotes adicionais:

- [OpenAI](#)
- [OpenAI do Azure](#)
- [Inferência de IA do Azure](#)
- [Olhamal](#)

No futuro, as implementações dessas `Microsoft.Extensions.AI` abstrações farão parte das respectivas bibliotecas de cliente, em vez de exigir a instalação de pacotes adicionais.

Implementações de middleware para serviços de IA

Conectar-se e usar serviços de IA é apenas um aspecto da criação de aplicativos robustos. Os aplicativos prontos para produção exigem recursos adicionais, como telemetria, registro em log e recursos de chamada de ferramentas. As `Microsoft.Extensions.AI` abstrações permitem que você integre facilmente esses componentes em seus aplicativos usando padrões familiares.

O exemplo a seguir demonstra como registrar um OpenAI `IChatClient`. `IChatClient` permite que você anexe os recursos de maneira consistente em vários provedores.

C#

```
app.Services.AddChatClient(builder => builder
    .UseLogging()
    .UseFunctionInvocation()
    .UseDistributedCache()
    .UseOpenTelemetry()
    .Use(new OpenAIClient(...)).AsChatClient(...));
```

Os recursos demonstrados neste snippet estão incluídos na `Microsoft.Extensions.AI` biblioteca, mas são apenas um pequeno subconjunto dos recursos que podem ser colocados em camadas com essa abordagem. Os desenvolvedores do .NET podem expor muitos tipos de middleware para criar uma funcionalidade de IA poderosa.

Crie com Microsoft.Extensions.AI

Você pode começar a criar com `Microsoft.Extensions.AI` das seguintes maneiras:

- **Desenvolvedores de biblioteca:** se você possui bibliotecas que fornecem clientes para serviços de IA, considere implementar as interfaces em suas bibliotecas. Isso permite que os usuários integrem facilmente seu pacote NuGet por meio das abstrações.
- **Consumidores de serviço:** se você estiver desenvolvendo bibliotecas que consomem serviços de IA, use as abstrações em vez de codificar para um serviço de IA específico. Essa abordagem oferece aos seus consumidores a flexibilidade de escolher seu serviço preferido.
- **Desenvolvedores de aplicativos:** use as abstrações para simplificar a integração em seus aplicativos. Isso permite a portabilidade entre modelos e serviços, facilita o teste e a simulação, aproveita o middleware fornecido pelo ecossistema e mantém uma API consistente em todo o aplicativo, mesmo que você use serviços diferentes em diferentes partes do aplicativo.
- **Colaboradores do ecossistema:** se você estiver interessado em contribuir para o ecossistema, considere escrever componentes de middleware personalizados. Para começar, confira os exemplos no [repositório GitHub dotnet/ai-samples](#).

Para obter um exemplo completo usando `Microsoft.Extensions.AI`, consulte [eShopSupport](#).

Próximas etapas

- [Criar um aplicativo de bate-papo de IA com o .NET](#)
- [Início rápido – resumir texto usando o aplicativo de chat de IA do Azure com o .NET](#)

Desenvolver aplicativos de IA com o .NET

Artigo • 18/06/2024

Este artigo contém uma lista organizada dos melhores recursos de aprendizado para desenvolvedores do .NET que estão começando a criar aplicativos de IA. Os recursos incluem artigos de início rápido populares, exemplos de referência, documentação e cursos de treinamento.

Recursos para o Serviço OpenAI do Azure

O Serviço OpenAI do Azure fornece acesso à API REST para modelos de linguagem avançados do OpenAI. Esses modelos podem ser facilmente adaptados à sua tarefa específica, incluindo, entre outros, geração de conteúdo, sumarização, reconhecimento de imagem, pesquisa semântica e tradução de linguagem natural para código. Os usuários podem acessar o serviço por meio de APIs REST, do SDK do OpenAI do Azure para .NET ou da interface baseada na Web no OpenAI Studio do Azure.

Bibliotecas e exemplos

[] Expandir a tabela

Link	Descrição
SDK do OpenAI do .NET ↗	A versão de origem do GitHub da biblioteca de clientes do OpenAI do Azure para .NET é uma adaptação das APIs REST do OpenAI que fornece uma interface idiomática e uma integração avançada com o restante do ecossistema do SDK do Azure. É possível conectar-se aos recursos do OpenAI do Azure ou ao ponto de extremidade de inferência do OpenAI que não-Azure, o que a torna uma excelente escolha até mesmo para desenvolvimento OpenAI não-Azure.
Versões do SDK do OpenAI do Azure ↗	Links para todos os pacotes de biblioteca do SDK do OpenAI do Azure, incluindo links para .NET, Java, JavaScript e Go.
Pacote NuGet Azure.AI.OpenAI ↗	A versão NuGet da biblioteca de clientes do OpenAI do Azure para .NET.
Introdução ao uso do GPT-35-Turbo e do GPT-4	Um artigo que orienta você através da criação de uma amostra de conclusão de chat.

Link	Descrição
Complementos ↗	Uma coleção de 10 amostras que demonstram como usar a biblioteca de clientes do OpenAI do Azure para .NET para conversar, transmitir respostas, usar seus próprios dados, transcrever/traduzir áudio, gerar imagens etc.
Transmissão de Conclusões de Chat ↗	Um link direto para as amostras que demonstram conclusões de streaming.
OpenAI com Controle de Acesso baseado em função do Microsoft Entra ID	Uma olhada na autenticação usando o Microsoft Entra ID.
OpenAI com Identidades Gerenciadas	Um artigo com cenários de segurança mais complexos que requerem o controle de acesso baseado em função do Azure (RBAC do Azure). Este documento aborda como autenticar em seu recurso OpenAI usando a Microsoft Entra ID.
Mais amostras ↗	Uma coleção de amostras do OpenAI gravadas em .NET.

Documentação

[\[+\] Expandir a tabela](#)

Link	Descrição
Documentação do Serviço OpenAI do Azure	A página do hub da documentação do Serviço OpenAI do Azure.
Visão geral do ecossistema .NET + AI	Resumo dos serviços e ferramentas que talvez seja necessário usar em seus aplicativos, com links para saber mais sobre cada um deles.
Criar um aplicativo de chat de IA do Azure com o .NET	Use o Kernel Semântico ou o SDK do OpenAI do Azure para criar um aplicativo de chat de console simples do .NET 8.
Resumir texto usando o aplicativo de chat de IA do Azure com o .NET	Similar ao artigo anterior, mas a solicitação é para resumir texto.
Obter informações sobre os seus dados de um aplicativo de chat de IA do Azure do .NET	Use o Kernel Semântico ou o SDK do OpenAI do Azure para obter análises e informações sobre seus dados.
Estender a IA do Azure usando ferramentas e executar uma função local com o .NET	Crie um assistente que lida com determinadas solicitações usando ferramentas personalizadas desenvolvidas em .NET.

Link	Descrição
Gerar imagens usando a IA do Azure com o .NET	Use o modelo OpenAI dell-e-3 para gerar uma imagem.

Recursos para outros Serviços de IA do Azure

Além do Serviço OpenAI do Azure, há muitos outros serviços de IA do Azure que ajudam desenvolvedores e organizações a criar rapidamente aplicativos inteligentes, prontos para o mercado e responsáveis, com APIs e modelos pré-construídos e personalizáveis. Os aplicativos de exemplo incluem processamento de idioma natural para conversas, pesquisa, monitoramento, tradução, fala, visão e tomada de decisão.

Amostras

 [Expandir a tabela](#)

Link	Descrição
Integrar a fala em seus aplicativos com Amostras de SDK de Fala	Um repositório de amostras para o SDK de Fala dos Serviços Cognitivos do Azure. Vinculados a amostras para reconhecimento de fala, tradução, síntese de fala e muito mais.
SDK de IA do Azure para Informação de Documentos	A IA do Azure para Informação de Documentos (anteriormente, Reconhecimento de Formulários) é um serviço de nuvem que usa aprendizado de máquina para analisar texto e dados estruturados de documentos. O SDK (kit de desenvolvimento de software) de Informação de Documentos é um conjunto de bibliotecas e ferramentas que permite integrar facilmente os recursos da Informação de Documentos em seus aplicativos.
Extrair dados estruturados de formulários, recibos, faturas e cartões usando o Reconhecimento de Formulários do Azure em .NET	Um repositório de amostras para a biblioteca de clientes Azure.IA.FormRecognizer.
Extrair, classificar e entender o texto em documentos usando a Análise de Texto em .NET	A Biblioteca de clientes para Análise de Texto. Isso faz parte do serviço Linguagem de IA do Azure , que fornece recursos de Processamento de Linguagem Natural (NLP) para reconhecimento e análise de texto.
Tradução de documentos em .NET	Um artigo de início rápido que detalha como usar a Tradução de Documentos para traduzir um documento de origem em um

Link	Descrição
	idioma de destino, preservando a estrutura e a formatação do texto.
Resposta às Perguntas em .NET ↗	Um artigo de início rápido para obter uma resposta (e pontuação de confiança) de um corpo de texto enviado junto com a pergunta.
Compreensão da Linguagem Coloquial em .NET ↗	A biblioteca de clientes para a Compreensão da Linguagem Coloquial (CLU), um serviço de IA conversacional baseado em nuvem, que pode extrair intenções e entidades em conversas e atua como um orquestrador para selecionar o melhor candidato para analisar conversas e obter a melhor resposta de aplicativos como Qna, Luis e Conversation App.
Analisa Imagens	Exemplo de código e documentos de configuração para o SDK de análise de imagem de IA do Microsoft Azure

Documentação

 Expandir a tabela

Serviço de IA	Descrição	Referência da API	Início rápido
Segurança do conteúdo	Um serviço de IA que detecta conteúdo indesejado.	Referência da API no Content Safety	Início rápido
Informação de documentos	Transforme documentos em soluções inteligentes baseadas em dados.	Referência da API do Document Intelligence	Início rápido
Idioma	Crie aplicativos com funcionalidades de compreensão de linguagem natural líderes do setor.	Referência da API de Linguagem	Início rápido
Pesquisar	Integre pesquisa na nuvem com IA aos seus aplicativos.	Referência da API de Pesquisa	Início rápido
Fala	Conversão de fala em texto, conversão de texto em fala, tradução e reconhecimento de locutor.	Referência da API de Fala	Início rápido
Tradutor	Use a tradução com IA para traduzir mais de 100 idiomas e dialetos em uso, em risco e ameaçados.	Referência da API de Tradução	Início rápido

Serviço de IA	Descrição	Referência da API	Início rápido
Serviço Cognitivo do Azure para Visão	Analisa o conteúdo em imagens e vídeos.	Referência da API de Visão	Início rápido

Treinamento

[\[+\] Expandir a tabela](#)

Link	Descrição
Workshop de IA generativa para iniciantes ↗	Conheça os conceitos básicos da criação de aplicativos de IA generativa com nosso curso abrangente de 18 lições do Microsoft Cloud Advocates.
Introdução aos Serviços de IA do Azure	Os Serviços de IA do Azure são uma coleção de serviços que são blocos construtores de IA que você pode integrar aos seus aplicativos. Neste roteiro de aprendizagem, você aprenderá a provisionar, proteger, monitorar e implantar recursos dos Serviços de IA do Azure e usá-los para criar soluções inteligentes.
Conceitos básicos de IA do Microsoft Azure: IA generativa	Caminho de treinamento para ajudá-lo a entender como os modelos de linguagem grandes formam a base da IA generativa: como o Serviço OpenAI do Azure fornece acesso à mais recente tecnologia de IA generativa, como solicitações e respostas podem ser ajustadas e como os princípios de IA responsável da Microsoft impulsionam os avanços éticos em IA.
Desenvolver soluções de IA generativa com o Serviço OpenAI do Azure	O Serviço OpenAI do Azure fornece acesso aos avançados modelos de linguagem grande do OpenAI, como modelos ChatGPT, GPT, Codex e Embeddings. Este roteiro de aprendizagem ensina os desenvolvedores a gerar código, imagens e texto usando o SDK do OpenAI do Azure e outros serviços do Azure.

Modelos de aplicativo de IA

Os modelos de aplicativo de IA fornecem implementações de referência regulares e fáceis de implantar que oferecem um ponto de partida de alta qualidade para os aplicativos de IA.

Há duas categorias de modelos de aplicativo de IA, **blocos de construção** e **soluções de ponta a ponta**. Blocos de construção são amostras em escala menor que focam em cenários e tarefas específicos. Soluções de ponta a ponta são amostras de referência

abrangentes, incluindo documentação, código-fonte e implantação para permitir executar e ampliar suas próprias finalidades.

Para examinar uma lista dos principais modelos disponíveis para cada linguagem de programação, consulte [Modelos de aplicativo de IA](#). Para procurar todos os modelos disponíveis, consulte os modelos de aplicativo de IA na [Galeria do Azure Developer CLI](#).

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).

.NET

Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 [Abrir um problema de documentação](#)

 [Fornecer comentários sobre o produto](#)

Visão geral do Semantic Kernel para .NET

Artigo • 04/10/2024

Neste artigo, explore os principais conceitos e recursos do [Semantic Kernel](#). O Kernel Semântico é uma opção avançada e recomendada para trabalhar com IA em aplicativos .NET. Nas próximas seções, você aprenderá:

- Como adicionar o Semantic Kernel ao seu projeto
- Conceitos centrais do Semantic Kernel

Este artigo apresenta uma visão geral introdutória do Kernel Semântico, especificamente no contexto do .NET. Para obter informações e treinamento mais abrangentes sobre o Semantic Kernel, consulte os recursos a seguir:

- [Documentação do Kernel Semântico](#)
- [Treinamento do Semantic Kernel](#)

Adicionar o Semantic Kernel a um projeto .NET

O SDK do Semantic Kernel está disponível como um pacote NuGet para .NET e se integra às configurações de aplicativo padrão.

Instale o pacote [Microsoft.SemanticKernel](#) usando o seguinte comando:

CLI do .NET

```
dotnet add package Microsoft.SemanticKernel
```

ⓘ Observação

Embora `Microsoft.SemanticKernel` forneça os principais recursos do Semantic Kernel, os recursos adicionais exigem a instalação de outros pacotes. Por exemplo, o pacote [Microsoft.SemanticKernel.Plugins.Memory](#) fornece recursos relacionados à memória de acesso. Para obter mais informações, consulte a documentação do [Semantic Kernel](#).

Crie e configure uma instância `Kernel` usando a classe `KernelBuilder` para acessar e trabalhar com o Semantic Kernel. O `Kernel` contém serviços, dados e conexões para orquestrar as integrações entre seu código e os modelos de IA.

Configure o Kernel em um aplicativo de console .NET:

```
C#  
  
var builder = Kernel.CreateBuilder();  
  
// Add builder configuration and services  
  
var kernel = builder.Build();
```

Configure o Kernel em um aplicativo ASP.NET Core:

```
C#  
  
var builder = WebApplication.CreateBuilder();  
builder.Services.AddKernel();  
  
// Add builder configuration and services  
  
var app = builder.Build();
```

Entenda o Semantic Kernel

Semantic Kernel é um SDK de código aberto que integra e orquestra modelos e serviços de IA como OpenAI, OpenAI do Azure e Hugging Face com linguagens de programação convencionais como C#, Python e Java.

O SDK do Semantic Kernel beneficia os desenvolvedores empresariais das seguintes maneiras:

- Simplifica a integração de recursos de IA em aplicativos existentes para permitir uma solução coesa para produtos empresariais.
- Minimiza a curva de aprendizado do trabalho com diferentes modelos ou serviços de IA, fornecendo abstrações que reduzem a complexidade.
- Aumenta a confiabilidade ao reduzir o comportamento imprevisível dos prompts e das respostas dos modelos de IA. É possível ajustar os prompts e planejar tarefas para criar uma experiência de usuário controlada e previsível.

O Semantic Kernel foi desenvolvido com base em vários conceitos fundamentais:

- **Conexões:** Interface com serviços externos de IA e fontes de dados.
- **Plug-ins:** encapsulam funções que os aplicativos podem usar.
- **Planner:** Orquestra planos e estratégias de execução com base no comportamento do usuário.

- **Memória:** Abstrai e simplifica o gerenciamento de contexto para aplicativos de IA.

Esses blocos de construção são explorados em mais detalhes nas seções a seguir.

conexões

O SDK do Semantic Kernel inclui um conjunto de conectores que permitem que os desenvolvedores integrem LLMs e outros serviços em seus aplicativos existentes. Esses conectores servem como ponte entre o código do aplicativo e os modelos ou serviços de IA. O Semantic Kernel lida com muitas preocupações e desafios comuns de conexão para que você possa se concentrar na criação de seus próprios fluxos de trabalho e recursos.

O trecho de código a seguir cria um `Kernel` e adiciona uma conexão a um modelo OpenAI do Azure:

```
C#  
  
using Microsoft.SemanticKernel;  
  
// Create kernel  
var builder = Kernel.CreateBuilder();  
  
// Add a chat completion service:  
builder.Services.AddAzureOpenAIChatCompletion(  
    "your-resource-name",  
    "your-endpoint",  
    "your-resource-key",  
    "deployment-model");  
var kernel = builder.Build();
```

Plug-ins

Os [plug-ins](#) do Semantic Kernel encapsulam as funções de linguagem padrão para aplicativos e modelos de IA consumirem. Você pode criar seus próprios plug-ins ou confiar nos plug-ins fornecidos pelo SDK. Esses plug-ins simplificam as tarefas em que os modelos de IA são vantajosos e os combinam eficientemente com métodos C# mais tradicionais. As funções de plug-in geralmente são categorizadas em dois tipos: *funções semânticas* e *funções nativas*.

Funções semânticas

As funções semânticas são essencialmente prompts de IA definidos em seu código que o Semantic Kernel pode personalizar e chamar conforme necessário. Você pode modelar

esses prompts para usar variáveis, formatação personalizada de prompts e conclusão, entre outros.

O trecho de código a seguir define e registra uma função semântica:

```
C#  
  
var userInput = Console.ReadLine();  
  
// Define semantic function inline.  
string skPrompt = @"Summarize the provided unstructured text in a sentence  
that is easy to understand.  
    Text to summarize: {{$userInput}}";  
  
// Register the function  
kernel.CreateSemanticFunction(  
    promptTemplate: skPrompt,  
    functionName: "SummarizeText",  
    pluginName: "SemanticFunctions"  
);
```

Funções nativas

As funções nativas são métodos C# que o Semantic Kernel pode chamar diretamente para manipular ou recuperar dados. Elas executam operações que são mais adequadas para instruções de código tradicionais em vez de prompts LLM.

O trecho de código a seguir define e registra uma função nativa:

```
C#  
  
// Define native function  
public class NativeFunctions {  
  
    [SKFunction, Description("Retrieve content from local file")]  
    public async Task<string> RetrieveLocalFile(string fileName, int maxSize  
= 5000)  
    {  
        string content = await File.ReadAllTextAsync(fileName);  
        if (content.Length <= maxSize) return content;  
        return content.Substring(0, maxSize);  
    }  
}  
  
//Import native function  
string plugInName = "NativeFunction";  
string functionName = "RetrieveLocalFile";
```

```
var nativeFunctions = new NativeFunctions();
kernel.ImportFunctions(nativeFunctions, plugInName);
```

Planner

O **Planner** é um componente central do Semantic Kernel que fornece orquestração de IA para gerenciar a integração perfeita entre modelos e plug-ins de IA. Essa camada cria estratégias de execução a partir de solicitações do usuário e orquestra dinamicamente os plug-ins para executar tarefas complexas com planejamento assistido por IA.

Considere o seguinte trecho do pseudocódigo:

C#

```
// Native function definition and kernel configuration code omitted for brevity

// Configure and create the plan
string planDefinition = "Read content from a local file and summarize the content.";
SequentialPlanner sequentialPlanner = new SequentialPlanner(kernel);

string assetsFolder = @"../../assets";
string fileName = Path.Combine(assetsFolder, "docs", "06_SemanticKernel",
"aci_documentation.txt");

ContextVariables contextVariables = new ContextVariables();
contextVariables.Add("fileName", fileName);

var customPlan = await sequentialPlanner.CreatePlanAsync(planDefinition);

// Execute the plan
KernelResult kernelResult = await kernel.RunAsync(contextVariables,
customPlan);
Console.WriteLine($"Summarization: {kernelResult.GetValue<string>()}");
```

O código anterior cria um plano executável e sequencial para ler o conteúdo de um arquivo local e resumir o conteúdo. O plano define instruções para ler o arquivo usando uma função nativa e, em seguida, analisá-lo usando um modelo de IA.

Memória

Os [repositórios de vetor](#) do Kernel Semântico fornecem abstrações sobre modelos de inserção, bancos de dados vetoriais e outros dados para simplificar o gerenciamento de contexto para aplicativos de IA. Os repositórios de vetor são agnósticos ao banco de dados LLM ou Vetor subjacente, proporcionando uma experiência de desenvolvedor

uniforme. Você pode configurar recursos de memória para armazenar os dados em diversas fontes ou serviços, incluindo a Pesquisa de IA do Azure e Cache do Azure para Redis.

Considere o seguinte snippet de código:

```
C#  
  
var facts = new Dictionary<string, string>();  
facts.Add(  
    "Azure Machine Learning; https://learn.microsoft.com/en-us/azure/machine-learning/",  
    @"Azure Machine Learning is a cloud service for accelerating and  
    managing the machine learning project lifecycle. Machine learning  
    professionals,  
    data scientists, and engineers can use it in their day-to-day workflows"  
);  
  
facts.Add(  
    "Azure SQL Service; https://learn.microsoft.com/en-us/azure/azure-sql/",  
    @"Azure SQL is a family of managed, secure, and intelligent products  
    that use the SQL Server database engine in the Azure cloud."  
);  
  
string memoryCollectionName = "SummarizedAzureDocs";  
  
foreach (var fact in facts) {  
    await memoryBuilder.SaveReferenceAsync(  
        collection: memoryCollectionName,  
        description: fact.Key.Split(";")[1].Trim(),  
        text: fact.Value,  
        externalId: fact.Key.Split(";")[2].Trim(),  
        externalSourceName: "Azure Documentation"  
    );  
}  
}
```

O código anterior carrega um conjunto de fatos na memória para que os dados estejam disponíveis para uso ao interagir com modelos de IA e orquestrar tarefas.

[Início Rápido: Resumir o texto com o OpenAI](#)

[Início Rápido: Conversar com seus dados](#)

Autenticação e autorização dos serviços de IA do Azure usando o .NET

Artigo • 08/07/2024

As solicitações de aplicativo para os Serviços de IA do Azure devem ser autenticadas. Neste artigo, você explorará as opções disponíveis para autenticar no OpenAI do Azure e em outros serviços de IA usando o .NET. Esses conceitos se aplicam ao SDK do Kernel Semântico, bem como SDKs de serviços específicos, como o OpenAI do Azure. A maioria dos serviços de IA oferece duas maneiras principais de autenticar aplicativos e usuários:

- **A autenticação baseada em chave** fornece acesso a um serviço do Azure usando valores de chave secreta. Esses valores secretos às vezes são conhecidos como chaves de API ou chaves de acesso, dependendo do serviço.
- **O Microsoft Entra ID** fornece uma solução abrangente de gerenciamento de identidade e acesso para garantir que as identidades corretas tenham o nível correto de acesso a diferentes recursos do Azure.

As seções adiante fornecem visões gerais conceituais para essas duas abordagens, em vez de etapas de implementação detalhadas. Para obter informações mais detalhadas sobre como se conectar aos serviços do Azure, visite os seguintes recursos:

- [Autenticar aplicativos .NET nos serviços do Azure](#)
- [Conceitos básicos de identidade](#)
- [O que é o Azure RBAC?](#)

ⓘ Observação

Os exemplos neste artigo se concentram principalmente em conexões com o OpenAI do Azure, mas os mesmos conceitos e etapas de implementação se aplicam diretamente a muitos outros serviços de IA do Azure também.

Autenticação usando chaves

As chaves de acesso permitem que aplicativos e ferramentas se autentiquem em um serviço de IA do Azure, como o OpenAI do Azure, usando uma chave secreta fornecida pelo serviço. Recupere a chave secreta usando ferramentas como o portal do Azure ou a CLI do Azure e use-a para configurar o código do aplicativo para se conectar ao serviço de IA:

C#

```
builder.Services.AddAzureOpenAIChatCompletion(
    "deployment-model",
    "service-endpoint",
    "service-key"); // Secret key
var kernel = builder.Build();
```

Usar chaves é uma opção simples, mas essa abordagem deve ser usada com cautela. As chaves não são a opção de autenticação recomendada porque elas:

- Não seguem [o princípio de privilégios mínimos](#). Elas fornecem permissões elevadas, independentemente de quem as usa ou para qual tarefa.
- Podem ser acionadas accidentalmente no controle do código-fonte ou armazenadas em locais não seguros.
- Podem ser facilmente compartilhadas ou enviadas para partes que não devem ter acesso.
- Geralmente, exigem administração e rotação manuais.

Em vez disso, considere usar o [Microsoft Entra ID](#) para autenticação, que é a solução recomendada para a maioria dos cenários.

Autenticação usando o Microsoft Entra ID

O Microsoft Entra ID é um serviço de gerenciamento de acesso e identidade baseado em nuvem que fornece um vasto conjunto de recursos para diferentes cenários de negócios e aplicativos. O Microsoft Entra ID é a solução recomendada para se conectar ao OpenAI do Azure e a outros serviços de IA e oferece os seguintes benefícios:

- Autenticação sem chave usando [identidades](#).
- RBAC (controle de acesso baseado em função) para atribuir às identidades as permissões mínimas necessárias.
- Pode usar a biblioteca de clientes [Azure.Identity](#) para detectar [credenciais diferentes entre ambientes](#) sem exigir alterações de código.
- Lida automaticamente com tarefas de manutenção administrativa, como girar chaves subjacentes.

O fluxo de trabalho para implementar a autenticação do Microsoft Entra em seu aplicativo geralmente inclui o seguinte:

- Desenvolvimento local:
 1. Entre no Azure usando uma ferramenta de desenvolvimento local, como a CLI do Azure ou o Visual Studio.

2. Configure seu código para usar a biblioteca do cliente [Azure.Identity](#) e a classe `DefaultAzureCredential`.
 3. Atribua funções do Azure à conta com a qual você entrou para habilitar o acesso ao serviço de IA.
- Aplicativo hospedado no Azure:
 1. Implante o aplicativo no Azure depois de configurá-lo para autenticar usando a biblioteca de clientes `Azure.Identity`.
 2. Atribua uma [identidade gerenciada](#) ao aplicativo hospedado pelo Azure.
 3. Atribua funções do Azure à identidade gerenciada para habilitar o acesso ao serviço de IA.

Os principais conceitos desse fluxo de trabalho são explorados nas seções a seguir.

Autenticar no Azure localmente

Ao desenvolver aplicativos localmente que se conectam aos serviços de IA do Azure, autentique-se no Azure usando uma ferramenta como o Visual Studio ou a CLI do Azure. Suas credenciais locais podem ser descobertas pela biblioteca de clientes `Azure.Identity` e usadas para autenticar seu aplicativo nos serviços do Azure, conforme descrito na seção [Configurar o código do aplicativo](#).

Por exemplo, para autenticar no Azure localmente usando a CLI do Azure, execute o seguinte comando:

```
CLI do Azure
az login
```

Configurar o código do aplicativo

Use a biblioteca de clientes [Azure.Identity](#) do SDK do Azure para implementar a autenticação do Microsoft Entra em seu código. As bibliotecas `Azure.Identity` incluem a classe `DefaultAzureCredential`, que descobre automaticamente as credenciais disponíveis do Azure com base no ambiente atual e nas ferramentas disponíveis. Visite a documentação do [SDK do Azure para .NET](#) para obter o conjunto completo de credenciais de ambiente com suporte e a ordem na qual elas são pesquisadas.

Por exemplo, configure o Kernel Semântico para autenticação usando `DefaultAzureCredential` com o seguinte código:

C#

```
Kernel kernel = Kernel
    .CreateBuilder()
    .AddAzureOpenAITextGeneration(
        "your-model",
        "your-endpoint",
        new DefaultAzureCredential())
    .Build();
```

`DefaultAzureCredential` permite que os aplicativos sejam promovidos do desenvolvimento local para a produção sem alterações de código. Por exemplo, durante o desenvolvimento `DefaultAzureCredential` usa suas credenciais de usuário local do Visual Studio ou da CLI do Azure para autenticar no serviço de IA. Quando o aplicativo é implantado no Azure, `DefaultAzureCredential` usa a identidade gerenciada atribuída ao seu aplicativo.

Atribuir funções à sua identidade

O RBAC (controle de acesso baseado em função) do Azure é um sistema que fornece gerenciamento de acesso refinado de recursos do Azure. Atribua uma função à entidade de segurança usada por `DefaultAzureCredential` para se conectar a um serviço de IA do Azure, seja um usuário individual, um grupo, uma entidade de serviço ou uma identidade gerenciada. As funções do Azure são uma coleção de permissões que permitem que a identidade execute várias tarefas, como gerar conclusões ou criar e excluir recursos.

Atribua funções como **Usuário OpenAI dos Serviços Cognitivos** (ID da função: `5e0bd9bd-7b93-4f28-af87-19fc36ad61bd`) à identidade relevante usando ferramentas como a CLI do Azure, o Bicep ou o Portal do Azure. Por exemplo, use o comando `az role assignment create` para atribuir uma função usando a CLI do Azure:

CLI do Azure

```
az role assignment create \
    --role "5e0bd9bd-7b93-4f28-af87-19fc36ad61bd" \
    --assignee-object-id "$PRINCIPAL_ID" \
    --scope
    /subscriptions/"$SUBSCRIPTION_ID"/resourceGroups/"$RESOURCE_GROUP" \
    --assignee-principal-type User
```

Saiba mais sobre o RBAC do Azure usando os seguintes recursos:

- O que é o Azure RBAC?

- Conceder acesso a um usuário
- Melhores práticas do RBAC

Atribuir uma identidade gerenciada ao seu aplicativo

Na maioria dos cenários, os aplicativos hospedados no Azure devem usar uma [identidade gerenciada](#) para se conectar a outros serviços, como o OpenAI do Azure. As identidades gerenciadas fornecem uma identidade totalmente gerenciada no Microsoft Entra ID para que os aplicativos usem ao se conectar a recursos que dão suporte à autenticação do Microsoft Entra. `DefaultAzureCredential` descobre a identidade associada ao seu aplicativo e a usa para se autenticar em outros serviços do Azure.

Há dois tipos de identidades gerenciadas que você pode atribuir ao seu aplicativo:

- Uma **identidade atribuída pelo sistema** é vinculada ao seu aplicativo e é excluída se o seu aplicativo for excluído. Um aplicativo só pode ter uma identidade atribuída pelo sistema.
- Uma **identidade atribuída pelo usuário** é um recurso independente do Azure que pode ser atribuído ao seu aplicativo. Um aplicativo pode ter várias identidades atribuídas pelo usuário.

Atribua funções a uma identidade gerenciada da mesma forma que você faria com uma conta de usuário individual, como a função de [usuário OpenAI dos Serviços Cognitivos](#). Saiba mais sobre como trabalhar com identidades gerenciadas usando os seguintes recursos:

- Visão geral de [identidades gerenciadas](#)
- Autenticar o Serviço de Aplicativo no OpenAI do Azure usando o Microsoft Entra ID
- Como usar [identidades gerenciadas para o Serviço de Aplicativo e o Azure Functions](#)



Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).



Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:



Abrir um problema de documentação

 Fornecer comentários sobre o
produto

Conectar-se e solicitar um modelo de IA usando o .NET

Artigo • 21/11/2024

Neste início rápido, você aprenderá a criar um aplicativo de chat do console do .NET para se conectar e solicitar um modelo OpenAI ou OpenAI do Azure. O aplicativo usa a [Microsoft.Extensions.AI](#) biblioteca para que você possa escrever código usando abstrações de IA em vez de um SDK específico. As abstrações de IA permitem que você altere o modelo de IA subjacente com alterações mínimas de código.

ⓘ Observação

A [Microsoft.Extensions.AI](#) biblioteca está atualmente em versão prévia.

Pré-requisitos

- SDK do .NET 8: [instalar o SDK do .NET 8](#).
- Uma assinatura do Azure – [Crie uma gratuitamente](#).
- Acesso ao [Serviço OpenAI do Azure](#).
- CLI do Desenvolvedor do Azure (opcional) – [instale ou atualize a CLI](#) do Desenvolvedor do Azure.

ⓘ Observação

Você também pode usar o [Kernel](#) Semântico para realizar as tarefas deste artigo. O Semantic Kernel é um SDK leve e de software livre que permite criar agentes de IA e integrar os modelos de IA mais recentes em seus aplicativos .NET.

Clone o repositório de amostra

Você pode criar seu próprio aplicativo e seguir as etapas nas seções a seguir ou clonar o repositório GitHub que contém os aplicativos de exemplo concluídos para todos os inícios rápidos. O repositório de exemplo também é estruturado como um modelo da CLI do Desenvolvedor do Azure que pode provisionar um recurso do OpenAI do Azure para você.

Bash

```
git clone https://github.com/dotnet/ai-samples.git
```

Criar o aplicativo

Conclua as etapas a seguir para criar um aplicativo de console .NET para se conectar a um modelo de IA.

1. Em um diretório vazio em seu computador, use o `dotnet new` comando para criar um novo aplicativo de console:

CLI do .NET

```
dotnet new console -o ExtensionsAI
```

2. Altere o diretório para a pasta do aplicativo:

CLI do .NET

```
cd ExtensionsAI
```

3. Instale os pacotes necessários:

Bash

```
dotnet add package OpenAI
dotnet add package Microsoft.Extensions.AI.OpenAI
dotnet add package Microsoft.Extensions.Configuration
dotnet add package Microsoft.Extensions.Configuration.UserSecrets
```

4. Abra o aplicativo no código do Visual Studio ou no editor de sua escolha

Bash

```
code .
```

Configurar o aplicativo

1. Navegue até a raiz do projeto .NET em um terminal ou prompt de comando.
2. Execute os seguintes comandos para configurar sua chave de API da OpenAI como um segredo para o aplicativo de exemplo:

Bash

```
dotnet user-secrets init
dotnet user-secrets set OpenAIKey <your-openai-key>
dotnet user-secrets set ModelName <your-openai-model-name>
```

Adicionar o código do aplicativo

O aplicativo usa o [Microsoft.Extensions.AI](#) pacote para enviar e receber solicitações para o modelo de IA.

1. No arquivo `Program.cs`, adicione o código a seguir para se conectar e autenticar no modelo de IA.

C#

```
using Microsoft.Extensions.AI;
using Microsoft.Extensions.Configuration;
using OpenAI;

var config = new ConfigurationBuilder().AddUserSecrets<Program>()
().Build();
string model = config["ModelName"];
string key = config["OpenAIKey"];

// Create the IChatClient
IChatClient client =
    new OpenAIClient(key).AsChatClient(model);
```

2. Leia o conteúdo do `benefits.md` arquivo e use-o para criar um prompt para o modelo. O prompt instrui o modelo a resumir o conteúdo de texto do arquivo.

C#

```
// Create and print out the prompt
string prompt = $"""
    summarize the the following text in 20 words or less:
{File.ReadAllText("benefits.md")}
""";
```



```
Console.WriteLine($"user >> {prompt}");
```

3. Chame a `InvokePromptAsync` função para enviar o prompt ao modelo para gerar uma resposta.

C#

```
// Submit the prompt and print out the response
ChatCompletion response = await client.CompleteAsync(prompt, new
    ChatOptions { MaxOutputTokens = 400 });
Console.WriteLine($"assistant >> {response}");
```

4. Use o comando `dotnet run` para executar o aplicativo:

CLI do .NET

```
dotnet run
```

O aplicativo imprime a resposta de conclusão do modelo de IA. Personalize o conteúdo de texto do `benefits.md` arquivo ou o comprimento do resumo para ver as diferenças nas respostas.

Próximas etapas

- Início Rápido – criar um aplicativo de chat de IA com o .NET
- Gerar texto e conversas com Conclusões do .NET e do OpenAI do Azure

Criar um aplicativo de chat alimentado por IA com o .NET

Artigo • 21/11/2024

Comece com a OpenAI e o [Kernel Semântico](#) criando um aplicativo de chat de console simples .NET 8. O aplicativo será executado localmente e usará o modelo `gpt-3.5-turbo` da OpenAI. Siga essas etapas para obter acesso à OpenAI e saiba como usar o Kernel Semântico.

Pré-requisitos

- SDK do .NET 8.0: [instalar o SDK do .NET 8.0](#).
- Uma [chave de API da OpenAI](#) para que você possa executar este exemplo.
- No Windows, o PowerShell `v7+` é necessário. Para validar sua versão, execute `pwsh` em um terminal. Deve retornar a versão atual. Se ele retornar um erro, execute o seguinte comando: `dotnet tool update --global PowerShell`.

Obter o projeto de exemplo

Clone o repositório de amostra

Você pode criar seu próprio aplicativo e seguir as etapas nas seções a seguir ou clonar o repositório GitHub que contém os aplicativos de exemplo concluídos para todos os inícios rápidos. O repositório de exemplo também é estruturado como um modelo da CLI do Desenvolvedor do Azure que pode provisionar um recurso do OpenAI do Azure para você.

Bash

```
git clone https://github.com/dotnet/ai-samples.git
```

Experimente a amostra HikerAI

1. Em um terminal ou prompt de comando, navegue até o diretório

```
src\quickstarts\azure-openai\semantic-kernel\02-HikerAI.
```

2. Execute os seguintes comandos para configurar sua chave de API da OpenAI como um segredo para o aplicativo de exemplo:

Bash

```
dotnet user-secrets init  
dotnet user-secrets set OpenAIKey <your-openai-key>
```

3. Use o comando `dotnet run` para executar o aplicativo:

CLI do .NET

```
dotnet run
```

Explore o código

O aplicativo usa o pacote [Microsoft.SemanticKernel](#) para enviar e receber solicitações para um serviço OpenAI.

O código do aplicativo está contido no arquivo `Program.cs`. As primeiras várias linhas de código definem valores de configuração e obtêm a Chave da OpenAI que foi definida anteriormente usando o comando `dotnet user-secrets`.

C#

```
var config = new ConfigurationBuilder().AddUserSecrets<Program>().Build();  
string model = "gpt-3.5-turbo";  
string key = config["OpenAIKey"];
```

O serviço `openAIChatCompletionService` facilita as solicitações e respostas.

C#

```
// Create the OpenAI Chat Completion Service  
OpenAIChatCompletionService service = new(model, key);
```

Adicione um prompt do sistema para fornecer mais contexto ao modelo, o que influencia o comportamento do modelo e as conclusões geradas durante a conversa.

C#

```
// Start the conversation with context for the AI model  
ChatHistory chatHistory = new("")  
    You are a hiking enthusiast who helps people discover fun hikes in their
```

area.

You are upbeat and friendly. You introduce yourself when first saying hello.

When helping people out, you always ask them for this information to inform the hiking recommendation you provide:

1. Where they are located
2. What hiking intensity they are looking for

You will then provide three suggestions for nearby hikes that vary in length

after you get that information. You will also share an interesting fact about

the local nature on the hikes when making a recommendation.
""");

Adicione uma mensagem de usuário ao histórico de chats usando a função `AddUserMessage`. Use a função `GetChatMessageContentAsync` para instruir o modelo a gerar uma resposta com base no prompt do sistema e na solicitação do usuário.

C#

```
// Add user message to chat history
chatHistory.AddUserMessage("Hi! Apparently you can help me find a hike that
I will like?");

// Print User Message to console
Console.WriteLine($"{chatHistory.Last().Role} >>>
{chatHistory.Last().Content}");

// Get response
var response = await service.GetChatMessageContentAsync(
    chatHistory, new OpenAIPromptExecutionSettings() { MaxTokens = 400 });
```

Adicione a resposta do modelo para manter o histórico de chat.

C#

```
// Add response to chat history
chatHistory.Add(response);

// Print Response to console
Console.WriteLine($"{chatHistory.Last().Role} >>>
{chatHistory.Last().Content}");
```

Personalize o prompt do sistema e a mensagem do usuário para ver como o modelo responde para ajudá-lo a encontrar um aumento que você desejará.

Próximas etapas

- Guia de início rápido: obter informações sobre seus dados do aplicativo de chat com IA do .NET
- Gerar texto e conversas com conclusões do .NET e da OpenAI

Obter informações sobre seus dados de um aplicativo de chat de IA do .NET

Artigo • 21/11/2024

Introdução ao desenvolvimento de IA usando um aplicativo de console do .NET 8 para se conectar a um modelo `gpt-3.5-turbo` do OpenAI. Você se conectará ao modelo de IA usando [Kernel Semântico](#) para analisar dados de caminhada e fornecer insights.

Pré-requisitos

- SDK do .NET 8.0: [instalar o SDK do .NET 8.0](#).
- Uma [chave de API da OpenAI](#) para que você possa executar este exemplo.
- No Windows, o PowerShell `v7+` é necessário. Para validar sua versão, execute `pwsh` em um terminal. Deve retornar a versão atual. Se ele retornar um erro, execute o seguinte comando: `dotnet tool update --global PowerShell`.

Obter o projeto de exemplo

Clone o repositório de amostra

Você pode criar seu próprio aplicativo e seguir as etapas nas seções a seguir ou clonar o repositório GitHub que contém os aplicativos de exemplo concluídos para todos os inícios rápidos. O repositório de exemplo também é estruturado como um modelo da CLI do Desenvolvedor do Azure que pode provisionar um recurso do OpenAI do Azure para você.

Bash

```
git clone https://github.com/dotnet/ai-samples.git
```

Experimentar o exemplo de chat de caminhada

1. Em um terminal ou prompt de comando, navegue até o diretório `src\quickstarts\openai\semantic-kernel\03-ChattingAboutMyHikes`.
2. Execute os seguintes comandos para configurar sua chave de API da OpenAI como um segredo para o aplicativo de exemplo:

Bash

```
dotnet user-secrets init  
dotnet user-secrets set OpenAIKey <your-openai-key>
```

3. Use o comando `dotnet run` para executar o aplicativo:

CLI do .NET

```
dotnet run
```

Explore o código

O aplicativo usa o pacote [Microsoft.SemanticKernel](#) para enviar e receber solicitações para um serviço OpenAI.

Todo o aplicativo está contido no arquivo `Program.cs`. As primeiras várias linhas de código definem valores de configuração e obtêm a Chave do OpenAI que foi definida anteriormente usando o comando `dotnet user-secrets`.

C#

```
var config = new ConfigurationBuilder().AddUserSecrets<Program>().Build();  
string model = "gpt-3.5-turbo";  
string key = config["OpenAIKey"];
```

O serviço `OpenAIChatCompletionService` facilita as solicitações e respostas.

C#

```
// Create the OpenAI Chat Completion Service  
OpenAIChatCompletionService service = new(model, key);
```

Depois que o cliente `OpenAIChatCompletionService` é criado, o aplicativo lê o conteúdo do arquivo `hikes.md` e o usa para fornecer mais contexto ao modelo adicionando um prompt do sistema. Isso influencia o comportamento do modelo e as conclusões geradas durante a conversa.

C#

```
// Provide context for the AI model  
ChatHistory chatHistory = new($"""  
    You are upbeat and friendly. You introduce yourself when first saying  
    hello.
```

Provide a short answer only based on the user hiking records below:

```
{File.ReadAllText("hikes.md")}  
""");  
Console.WriteLine($"{chatHistory.Last().Role} >>>  
{chatHistory.Last().Content}");
```

O código a seguir adiciona um prompt de usuário ao modelo usando a função `AddUserMessage`. A função `GetChatMessageContentAsync` instrui o modelo a gerar uma resposta com base nos prompts do sistema e do usuário.

C#

```
// Start the conversation  
chatHistory.AddUserMessage("Hi!");  
Console.WriteLine($"{chatHistory.Last().Role} >>>  
{chatHistory.Last().Content}");  
  
chatHistory.Add(  
    await service.GetChatMessageContentAsync(  
        chatHistory,  
        new OpenAIPromptExecutionSettings()  
    {  
        MaxTokens = 400  
    }));  
Console.WriteLine($"{chatHistory.Last().Role} >>>  
{chatHistory.Last().Content}");
```

O aplicativo adiciona a resposta do modelo ao `chatHistory` para manter o histórico ou contexto do chat.

C#

```
// Continue the conversation with a question.  
chatHistory.AddUserMessage(  
    "I would like to know the ratio of the hikes I've done in Canada  
    compared to other countries.");  
  
Console.WriteLine($"{chatHistory.Last().Role} >>>  
{chatHistory.Last().Content}");  
  
chatHistory.Add(await service.GetChatMessageContentAsync(  
    chatHistory,  
    new OpenAIPromptExecutionSettings()  
    {  
        MaxTokens = 400  
    }));
```

```
Console.WriteLine($"{chatHistory.Last().Role} >>>  
{chatHistory.Last().Content}");
```

Personalize os prompts do sistema ou do usuário para fornecer perguntas e contexto diferentes:

- Quantas vezes eu caminhei quando estava chovendo?
- Quantas vezes eu caminhei em 2021?

O modelo gera uma resposta relevante para cada prompt com base em suas entradas.

Próximas etapas

- Início rápido – Gerar imagens usando IA com .NET
- Gerar texto e conversas com Conclusões do .NET e do OpenAI do Azure

Estender o OpenAI usando Ferramentas e executar uma função local com o .NET

Artigo • 21/11/2024

Comece a usar a IA criando um aplicativo de chat de console simples do .NET 8. O aplicativo será executado localmente e usará o modelo OpenAI `gpt-3.5-turbo`, usando ferramentas para estender os recursos do modelo chamando um método .NET local. Siga essas etapas para obter acesso ao OpenAI e saiba como usar o Kernel Semântico.

Pré-requisitos

- SDK do .NET 8.0: [instalar o SDK do .NET 8.0 ↗](#).
- Uma [chave de API da OpenAI ↗](#) para que você possa executar este exemplo.
- No Windows, o PowerShell `v7+` é necessário. Para validar sua versão, execute `pwsh` em um terminal. Deve retornar a versão atual. Se ele retornar um erro, execute o seguinte comando: `dotnet tool update --global PowerShell`.

Obter o projeto de exemplo

Clone o repositório de amostra

Você pode criar seu próprio aplicativo e seguir as etapas nas seções a seguir ou clonar o repositório GitHub que contém os aplicativos de exemplo concluídos para todos os inícios rápidos. O repositório de exemplo também é estruturado como um modelo da CLI do Desenvolvedor do Azure que pode provisionar um recurso do OpenAI do Azure para você.

Bash

```
git clone https://github.com/dotnet/ai-samples.git
```

Experimente o exemplo hiker pro

1. Em um terminal ou prompt de comando, navegue até o diretório `src\quickstarts\openai\semantic-kernel\04-HikerAIPro`.

2. Execute os seguintes comandos para configurar sua chave de API da OpenAI como um segredo para o aplicativo de exemplo:

Bash

```
dotnet user-secrets init  
dotnet user-secrets set OpenAIKey <your-openai-key>
```

3. Use o comando `dotnet run` para executar o aplicativo:

CLI do .NET

```
dotnet run
```

Compreender o código

O aplicativo usa o pacote [Microsoft.SemanticKernel](#) para enviar e receber solicitações para um serviço OpenAI.

Todo o aplicativo está contido no arquivo `Program.cs`. As primeiras várias linhas de código definem valores de configuração e obtêm a Chave do OpenAI que foi definida anteriormente usando o comando `dotnet user-secrets`.

C#

```
var config = new ConfigurationBuilder().AddUserSecrets<Program>().Build();  
string model = "gpt-3.5-turbo";  
string key = config["OpenAIKey"];
```

A classe `Kernel` facilita as solicitações e respostas com a ajuda do serviço `AddOpenAIChatCompletion`.

C#

```
// Create a Kernel containing the OpenAI Chat Completion Service  
IKernelBuilder b = Kernel.CreateBuilder();  
  
Kernel kernel = b  
    .AddOpenAIChatCompletion(model, key)  
    .Build();
```

As funções `ImportPluginFromFunctions` e `CreateFromMethod` são usadas para definir a função local que será chamada pelo modelo.

C#

```
// Add a new plugin with a local .NET function that should be available to
// the AI model
// For convenience and clarity of into the code, this standalone local
// method handles tool call responses. It will fake a call to a weather API and
// return the current weather for the specified location.
kernel.ImportPluginFromFunctions("WeatherPlugin",
[
    KernelFunctionFactory.CreateFromMethod(
        [Description("The city, e.g. Montreal, Sidney")] string location,
        string unit = null) =>
    {
        // Here you would call a weather API to get the weather for the
        location
        return "Periods of rain or drizzle, 15 C";
    }, "get_current_weather", "Get the current weather in a given location")
]);
```

Depois que o `kernel` cliente é criado, o código usa um prompt do sistema para fornecer contexto e influenciar o tom de conclusão e o conteúdo. Observe como o clima é enfatizado no prompt do sistema.

C#

```
ChatHistory chatHistory = new"""
    You are a hiking enthusiast who helps people discover fun hikes in their
area.
    You are upbeat and friendly. Good weather is important for a good hike.
    Only make recommendations if the weather is good or if people insist.
    You introduce yourself when first saying hello. When helping people out,
    you always ask them for this information to inform the hiking
recommendation you provide:

    1. Where they are located
    2. What hiking intensity they are looking for

    You will then provide three suggestions for nearby hikes that vary in
length
    after you get that information. You will also share an interesting fact
about the local
    nature on the hikes when making a recommendation.
""");
```

O aplicativo também adiciona uma mensagem de usuário ao modelo usando a função `AddUserMessage`. A função `GetChatMessageContentAsync` envia o histórico de chats para o modelo a gerar uma resposta com base nos prompts do sistema e do usuário.

C#

```
chatHistory.AddUserMessage("""
    Is the weather is good today for a hike?
    If yes, I live in the greater Montreal area and would like an easy hike.
    I don't mind driving a bit to get there. I don't want the hike to be
over 10 miles round trip.
    I'd consider a point-to-point hike.
    I want the hike to be as isolated as possible. I don't want to see many
people.
    I would like it to be as bug free as possible.
""");

Console.WriteLine($"{chatHistory.Last().Role} >>>
{chatHistory.Last().Content}");

chatHistory.Add(await service.GetChatMessageContentAsync(
    chatHistory,
    new OpenAIPromptExecutionSettings()
{
    MaxTokens = 400
}));

Console.WriteLine($"{chatHistory.Last().Role} >>>
{chatHistory.Last().Content}");
```

Personalize o prompt do sistema e a mensagem do usuário para ver como o modelo responde para ajudá-lo a encontrar um aumento que você desejará.

Próximas etapas

- Início rápido – Obter informações sobre seus dados de um aplicativo de chat de IA do .NET
- Gerar texto e conversas com Conclusões do .NET e do OpenAI do Azure

Gerar imagens usando a IA com o .NET

Artigo • 21/11/2024

Comece a usar a IA criando um aplicativo de chat de console simples do .NET 8. O aplicativo será executado localmente e usará o modelo `dall-e-3` do OpenAI para gerar imagens de cartão postal para que você possa convidar seus amigos para uma caminhada. Siga essas etapas para obter acesso ao OpenAI e saiba como usar o Kernel Semântico.

Pré-requisitos

- SDK do .NET 8.0: [instalar o SDK do .NET 8.0 ↗](#).
- Uma [chave de API da OpenAI ↗](#) para que você possa executar este exemplo.
- No Windows, o PowerShell `v7+` é necessário. Para validar sua versão, execute `pwsh` em um terminal. Deve retornar a versão atual. Se ele retornar um erro, execute o seguinte comando: `dotnet tool update --global PowerShell`.

Obter o projeto de exemplo

Clone o repositório de amostra

Você pode criar seu próprio aplicativo e seguir as etapas nas seções a seguir ou clonar o repositório GitHub que contém os aplicativos de exemplo concluídos para todos os inícios rápidos. O repositório de exemplo também é estruturado como um modelo da CLI do Desenvolvedor do Azure que pode provisionar um recurso do OpenAI do Azure para você.

Bash

```
git clone https://github.com/dotnet/ai-samples.git
```

Experimentar o exemplo de imagens de caminhada

1. Clone o repositório: [dotnet/ai-samples ↗](#)
2. Execute os seguintes comandos para configurar sua chave de API do OpenAI como um segredo para o aplicativo de exemplo:

Bash

```
dotnet user-secrets init  
dotnet user-secrets set OpenAIKey <your-openai-key>
```

3. Use o comando `dotnet run` para executar o aplicativo:

CLI do .NET

```
dotnet run
```

Explore o código

O aplicativo usa o pacote [Microsoft.SemanticKernel](#) para enviar e receber solicitações para o serviço OpenAI.

O arquivo `Program.cs` contém todo o código do aplicativo. As primeiras várias linhas de código definem valores de configuração e obtêm a Chave do OpenAI que foi definida anteriormente usando o comando `dotnet user-secrets`.

C#

```
var config = new ConfigurationBuilder().AddUserSecrets<Program>().Build();  
string key = config["OpenAIKey"];
```

O serviço `openAITextToImageService` facilita as solicitações e respostas.

C#

```
OpenAITextToImageService textToImageService = new(key, null);
```

Forneça contexto e instruções ao modelo adicionando um prompt do sistema. Um bom prompt de geração de imagem requer uma descrição clara do que é a imagem, quais cores usar, o estilo pretendido e outros descritores.

A função `GenerateImageAsync` instrui o modelo a gerar uma resposta com base no prompt do usuário e nas configurações de qualidade e tamanho da imagem.

C#

```
// Generate the image  
string imageUrl = await textToImageService.GenerateImageAsync("")  
A postal card with a happy hiker waving and a beautiful mountain in the
```

```
background.  
There is a trail visible in the foreground.  
The postal card has text in red saying: 'You are invited for a hike!'  
"""", 1024, 1024);  
  
Console.WriteLine($"The generated image is ready at:\n{imageUrl}");
```

Personalize o prompt para personalizar as imagens geradas pelo modelo.

Próximas etapas

- Início rápido – resumir texto usando o aplicativo de chat de IA com o .NET
- Gerar texto e conversas com Conclusões do .NET e do OpenAI do Azure

Conversar com um modelo de IA local usando o Kernel Semântico e o .NET

Artigo • 24/11/2024

Os modelos locais de IA fornecem opções avançadas e flexíveis para criar soluções de IA. Neste início rápido, você explorará como configurar e se conectar a um modelo de IA local usando o .NET e o SDK do Kernel Semântico. Para este exemplo, você executará o modelo de IA local usando o Ollama.

Pré-requisitos

- [Instalar o .NET 8.0](#) ou superior
- [instalar o Ollama](#) localmente em seu dispositivo
- [Visual Studio Code](#) (opcional)

Executar o modelo de IA local

Conclua as etapas a seguir para configurar e executar um modelo de IA local em seu dispositivo. Muitos modelos de IA diferentes estão disponíveis para execução local e são treinados para tarefas diferentes, como gerar código, analisar imagens, chat gerativo ou criar inserções. Para este início rápido, você usará o modelo `phi3:mini` da finalidade geral, que é uma IA generativa pequena, mas capaz, criada pela Microsoft.

1. Abra uma janela do terminal e verifique se o Ollama está disponível em seu dispositivo:

```
Bash
```

```
ollama
```

Se o Ollama estiver em execução, ele exibirá uma lista de comandos disponíveis.

2. Efetue pull do modelo `phi3:mini` do Registro Ollama e aguarde até que ele seja baixado:

```
Bash
```

```
ollama pull phi3:mini
```

3. Após a conclusão do download, execute o modelo:

```
Bash
```

```
ollama run phi3:mini
```

O Ollama inicia o modelo `phi3:mini` e fornece um prompt para você interagir com ele.

Criar o aplicativo .NET

Conclua as seguintes etapas para criar um aplicativo de console do .NET que se conectará ao modelo de IA `phi3:mini` local:

1. Em uma janela de terminal, navegue até um diretório vazio em seu dispositivo e crie um novo aplicativo com o comando `dotnet new`:

```
CLI do .NET
```

```
dotnet new console
```

2. Adicione o SDK [do Kernel Semântico](#) e os pacotes do Conector Ollama [do Kernel Semântico](#) ao seu aplicativo:

```
CLI do .NET
```

```
dotnet add package Microsoft.SemanticKernel  
dotnet add package Microsoft.SemanticKernel.Connectors.Ollama
```

3. Abra o novo aplicativo em seu editor de escolha, como o Visual Studio Code.

```
CLI do .NET
```

```
code .
```

Conectar-se e conversar com o modelo de IA

O SDK do Kernel Semântico fornece muitos serviços e recursos para se conectar a modelos de IA e gerenciar interações. Nas etapas a seguir, você criará um aplicativo simples que se conectará à IA local e armazenará o histórico de conversas para melhorar a experiência de chat.

1. Abra o arquivo *Program.cs* e substitua o conteúdo do arquivo pelo seguinte código:

```
C#  
  
using Microsoft.SemanticKernel;  
using Microsoft.SemanticKernel.ChatCompletion;  
  
// Create a kernel with OpenAI chat completion  
// Warning due to the experimental state of some Semantic Kernel SDK  
// features.  
#pragma warning disable SKEXP0070  
Kernel kernel = Kernel.CreateBuilder()  
    .AddOllamaChatCompletion(  
        modelId: "phi3:mini",  
        endpoint: new Uri("http://localhost:11434"))  
    .Build();  
  
var aiChatService = kernel.GetRequiredService<IChatCompletionService>()  
();  
var chatHistory = new ChatHistory();  
  
while (true)  
{  
    // Get user prompt and add to chat history  
    Console.WriteLine("Your prompt:");  
    var userPrompt = Console.ReadLine();  
    chatHistory.Add(new ChatMessageContent(AuthorRole.User,  
userPrompt));  
  
    // Stream the AI response and add to chat history  
    Console.WriteLine("AI Response:");  
    var response = "";  
    await foreach(var item in  
  
aiChatService.GetStreamingChatMessageContentsAsync(chatHistory))  
{  
    Console.Write(item.Content);  
    response += item.Content;  
}  
    chatHistory.Add(new ChatMessageContent(AuthorRole.Assistant,  
response));  
    Console.WriteLine();  
}
```

O código anterior realiza as seguintes tarefas:

- Cria um objeto `Kernel` e o usa para recuperar um serviço de conclusão de chat.
- Cria um objeto `chatHistory` para armazenar as mensagens entre o usuário e o modelo de IA.

- Recupera um prompt do usuário e o armazena no `ChatHistory`.
- Envia os dados de chat para o modelo de IA para gerar uma resposta.

① Observação

O Ollama é executado na porta 11434 por padrão, e é por isso que o ponto de extremidade do modelo de IA é definido como `http://localhost:11434`.

2. Execute o aplicativo e insira um prompt no console para receber uma resposta da IA, como o seguinte:

Saída

Your prompt:
Tell me three facts about .NET.

AI response:

1. **Cross-Platform Development:** One of the significant strengths of .NET, particularly its newer iterations (.NET Core and .NET 5+), is cross-platform support. It allows developers to build applications that run on Windows, Linux, macOS, and various other operating systems seamlessly, enhancing flexibility and reducing barriers for a wider range of users.

2. **Rich Ecosystem and Library Support:** .NET has an incredibly rich ecosystem, comprising an extensive collection of libraries (such as those provided by the official NuGet Package Manager), tools, and services. This allows developers to work on web applications (.NET Framework for desktop apps and ASP.NET Core for modern web applications), mobile applications (.NET MAUI or Xamarin.Forms), IoT solutions, AI/ML projects, and much more with a vast array of pre-built components available at their disposal.

3. **Type Safety:** .NET operates under the Common Language Infrastructure (CLI) model and employs managed code for executing applications. This approach inherently offers strong type safety checks which help in preventing many runtime errors that are common in languages like C/C++. It also enables features such as garbage collection, thus relieving developers from manual memory management. These

characteristics enhance the reliability of .NET-developed software and improve productivity by catching issues early during development.

3. A resposta da IA é precisa, mas também detalhada. O histórico de chat armazenado permite que a IA modifique sua resposta. Instrua a IA a reduzir a lista fornecida por ela:

Saída

Your prompt:

Shorten the length of each item in the previous response.

AI Response:

****Cross-platform Capabilities:**** .NET allows building for various operating systems through platforms like .NET Core, promoting accessibility (Windows, Linux, macOS).

****Extensive Ecosystem:**** Offers a vast library selection via NuGet and tools for web (.NET Framework), mobile development (Maui/Xamarin.Forms), IoT, AI, providing rich capabilities to developers.

****Type Safety & Reliability:**** .NET's CLI model enforces strong typing and automatic garbage collection, mitigating runtime errors, thus enhancing application stability.

A resposta atualizada da IA é muito menor na segunda vez. Devido ao histórico de chat disponível, a IA conseguiu avaliar o resultado anterior e fornecer resumos mais curtos.

Próximas etapas

- Início Rápido – obter informações sobre seus dados de um aplicativo de chat de IA do Azure do .NET
- Gerar texto e conversas com conclusões do .NET e do OpenAI do Azure

Como a IA e os LLMs generativos funcionam

Artigo • 24/11/2024

A IA gerativa é um tipo de inteligência artificial capaz de criar conteúdo original, como linguagem natural, imagens, áudio e código. A saída de uma IA gerativa baseia-se nas entradas fornecidas pelo usuário. Uma maneira comum dos usuários interagirem com a IA gerativa é por meio de aplicativos de chat que usam a linguagem natural como entrada. O ChatGPT, desenvolvido pela OpenAI, é um exemplo popular disso. Aplicativos de IA gerativos que usam linguagem natural como entrada são alimentados por LLMs (modelos de linguagem grandes) para executar o NLP (processamento de linguagem natural).

Como a IA gerativa funciona

Toda IA gerativa é criada com base em modelos. Esses modelos são treinados com grandes conjuntos de dados na forma de conteúdo, como linguagem natural, imagens, áudio e código. Os modelos de IA gerativos usam os padrões identificados nos dados de treinamento para produzir conteúdo novo e estatisticamente semelhante.

A entrada fornecida pelo usuário é usada pelo modelo de IA para criar uma saída. A entrada é analisada primeiro em uma forma de dados que o modelo pode entender. Em seguida, o modelo usa esses dados para identificar padrões correspondentes de seu treinamento que ele combina para criar a saída final. Os modelos de IA gerativos são projetados para produzir conteúdo exclusivo, portanto, eles não gerarão a mesma saída para entradas idênticas.

Aplicativos de IA gerativos que dão suporte a linguagem natural como entrada ou saída utilizam LLMs para fazer isso. O LLM é usado para executar o NLP, que classifica o texto de entrada e determina seu sentimento. Essa análise de classificação e sentimento é usada pelo modelo de IA gerativo para identificar padrões e criar a saída. Se a saída for texto, o LLM sozinho poderá ser usado para gerá-lo. Se a saída for áudio ou imagens, modelos adicionais serão usados para fornecer os dados e padrões para gerar saídas nesse formato.

Usos comuns da IA gerativa

Os aplicativos de IA gerativa dão suporte a uma variedade de possíveis casos de uso e possíveis saídas, que são exploradas nas seções a seguir.

Geração de linguagem natural

Alguns aplicativos de IA generativos podem produzir saídas de linguagem natural, como:

- Resumos redigidos profissionalmente para um perfil do LinkedIn
- Descrições avançadas de itens em uma vitrine online
- Ideias para refeições saudáveis

Por exemplo, dada a entrada do usuário *Me dê três ideias para um café da manhã saudável que inclua pimenta*, um aplicativo de IA generativo pode produzir a seguinte saída:

Ok, aqui estão três cafés da manhã saudáveis que incluem pimentas:

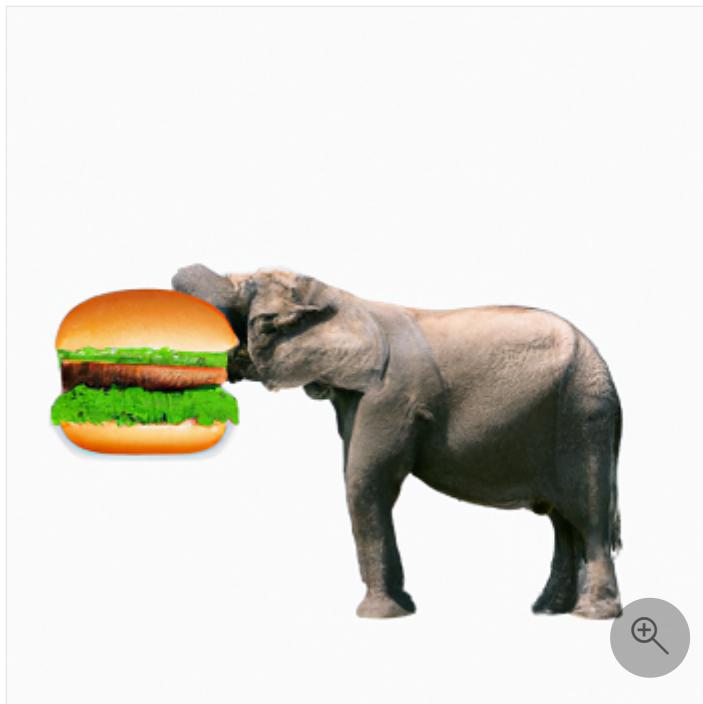
1. *Ovo mexido com pimenta: pique alguns pimentões e refogue-os em uma panela até que eles amoleçam ligeiramente. Em seguida, adicione alguns ovos batidos e misture tudo. Sirva com torrada integral para um café da manhã nutritivo.*
2. *Torrada de abacate com pimenta: torre uma fatia de pão integral e cubra-o com pimentões fatiados, abacate amassado e um pouco de sal e pimenta. Esta é uma ótima maneira de obter algumas gorduras saudáveis do abacate junto com a vitamina C das pimentas.*
3. *Burrito com pimenta: refogue pimentão picado, cebola e ovos mexidos em uma panela. Em seguida, use uma colher para colocar mistura em uma tortilha integral e finalize com um pouco de molho ou molho picante. Enrole e desfrute de um café da manhã satisfatório e nutritivo.*

Geração de imagem

Alguns aplicativos de IA generativa produzem saídas de imagem de entradas em linguagem natural. Dependendo do aplicativo e de seus modelos, os usuários podem ser capazes de especificar:

- As pessoas ou as coisas a serem exibidas na imagem
- O estilo artístico para criação da imagem
- Referências para gerar imagens semelhantes

A geração de imagens pode criar avatares virtuais para contas online, logotipos de design para uma empresa ou fornecer inspiração artística para criadores. Por exemplo, um usuário pode inserir a solicitação, *Crie uma imagem de um elefante comendo um hambúrguer*. Um aplicativo de IA gerativa pode produzir a seguinte saída:



Geração de áudio

Alguns aplicativos de IA generativa produzem saídas de áudio de entradas em linguagem natural. Dependendo do aplicativo e de seus modelos, os usuários podem ser capazes de:

- Sintetizar vozes de som natural do texto de entrada
- Criar música em um estilo específico ou com determinados instrumentos
- Modificar arquivos de áudio de entrada com base em um conjunto de critérios fornecidos em linguagem natural

A geração de áudio pode fornecer respostas faladas de assistentes de voz digital, adicionar instrumentos de apoio a músicas para produção musical ou reproduzir a voz original de um usuário de gravações de referência.

Geração de código

Alguns aplicativos de IA generativa produzem código de entradas em linguagem natural. Dependendo do aplicativo e de seus modelos, os usuários podem ser capazes de gerar código que:

- Esteja no idioma de sua escolha
- Aceite uma estrutura de entrada e saída específica
- Use tecnologias específicas com base na documentação de referência

A geração de código pode fornecer implementações rápidas de operações comuns, como classificar ou filtrar dados, acelerar a tradução de código de uma linguagem para

outra ou responder a perguntas sobre uma tecnologia com base na documentação de referência especificada. Por exemplo, considerando a entrada "Me mostre como codificar um jogo da velha com C#", um aplicativo de IA gerativa pode produzir a seguinte saída:

Aqui está um exemplo simples de como você pode codificar um jogo da velha em C#:

```
C#  
  
using System;  
  
namespace TicTacToe  
{  
    class Program  
    {  
        static char[] board = { '1', '2', '3', '4', '5', '6', '7', '8', '9' };  
        static int player = 1; // Player 1 will start  
        static int choice; // The number the player chooses  
        static int flag = 0; // Set the flag value to 0  
  
        // Check if the position is already marked or not  
        private static int CheckPosition(char mark)  
        {  
            for (int i = 0; i < 9; i++)  
            {  
                if (board[i] == mark)  
                {  
                    return 1;  
                }  
            }  
            return 0;  
        }  
  
        // The rest of the generated code has been omitted for brevity  
        // ...  
    }  
}
```

Esse código cria um jogo da velha simples baseado em console em C#. Ele usa uma matriz unidimensional para representar o tabuleiro e verifica se há uma vitória ou um empate após cada movimento.

Como os LLMs funcionam

Ao treinar um LLM, o texto de treinamento é dividido pela primeira vez em **tokens**. Cada token identifica um valor de texto exclusivo. Um token pode ser uma palavra distinta, uma palavra parcial ou uma combinação de palavras e pontuação. Cada token recebe

uma ID, que permite que o texto seja representado como uma sequência de IDs de token.

Depois que o texto for dividido em tokens, um vetor contextual, conhecido como [inserção](#), será atribuído a cada token. Esses vetores de inserção são dados numéricos de vários valores em que cada elemento do vetor de um token representa um atributo semântico do token. Os elementos do vetor de um token são determinados com base na frequência com que os tokens são usados juntos ou em contextos semelhantes.

A meta é prever o próximo token na sequência com base nos tokens anteriores. Um peso é atribuído a cada token na sequência existente que representa sua influência relativa no próximo token. Em seguida, é realizado um cálculo que usa os pesos e as inserções dos tokens anteriores para prever o próximo valor de vetor. Em seguida, o modelo seleciona o token mais provável para continuar a sequência com base no vetor previsto.

Esse processo continua iterativamente para cada token na sequência, com a sequência de saída sendo usada regressivamente como a entrada para a próxima iteração. A saída é criada um token de cada vez. Essa estratégia é análoga à forma como a conclusão automática funciona, em que as sugestões são baseadas no que foi digitado até agora e atualizado com cada nova entrada.

Durante o treinamento, a sequência completa de tokens é conhecida, mas todos os tokens que vêm após o que está sendo considerado atualmente são ignorados. O valor previsto para o vetor do próximo token é comparado ao valor real e a perda é calculada. Em seguida, os pesos são ajustados incrementalmente para reduzir a perda e aprimorar o modelo.

Conteúdo relacionado

- [Noções básicas de tokens](#)
- [Engenharia de prompts](#)
- [Modelos de linguagem grandes](#)

Como os agentes e copilotos trabalham com os LLMs

Artigo • 24/11/2024

Tanto os agentes quanto os copilotos ampliam os recursos de um LLM invocando de forma inteligente a funcionalidade externa, como o envio de um email.

- Um *agente* é uma inteligência artificial que pode responder a perguntas e automatizar processos para os usuários. Os agentes podem determinar quais funções atenderão ao objetivo de um usuário e, em seguida, chamar essas funções em nome do usuário.
- Um *copiloto* é um tipo de agente que trabalha lado a lado com um usuário. Ao contrário de um agente, um copiloto não é totalmente automatizado, ele depende da interação do usuário. Um copiloto pode ajudar um usuário a concluir uma tarefa fornecendo sugestões e recomendações.

Por exemplo, suponha que você esteja criando um aplicativo auxiliar de chat por email. Junto com o LLM, você também precisará de um plug-in para executar ações relacionadas a email, bem como plug-ins para pesquisa, resumo, determinação de intenção e similares. Você pode usar [funções nativas](#), [plug-ins prontos para uso](#) e seus próprios [plug-ins personalizados](#).

Criar os plug-ins é apenas metade da batalha: você ainda precisa invocar as funções certas no momento certo, um processo que pode ser propenso a erros e ineficiente. Um agente pode lidar melhor com isso.

Um agente decide automaticamente qual sequência de funções um LLM pode usar para atingir uma meta do usuário. Por exemplo, suponha que você tenha um aplicativo de chat que analisa os novos itens da caixa de entrada e determina a ação necessária para cada item. Se você configurar um agente, ele poderá orquestrar as funções de plug-in necessárias e executar as etapas automaticamente.

Componentes de um agente

Cada agente tem três blocos de construção principais: uma persona, plug-ins e planners.

- [Personas](#) determinam a maneira pela qual os agentes respondem aos usuários ou executam ações.

- [Plug-ins](#) permitem que os agentes recuperem informações do usuário ou de outros sistemas. Você pode usar plug-ins pré-criados e seus próprios plug-ins personalizados.
- [Planners](#) permitem que os agentes planejem como usar os plug-ins disponíveis.

Personas

A persona de um agente é sua identidade: todos os plug-ins e planners que o agente usa são ferramentas, mas a persona determina como ele usa essas ferramentas. Use [instruções](#) em um prompt para estabelecer a persona de um agente.

Por exemplo, você pode usar instruções para dizer a um agente que ele está ajudando as pessoas a gerenciar emails e para explicar suas decisões à medida que as toma. Seu prompt pode ser mais ou menos assim:

```
C#
```

```
prompt = $"""
    <message role="system">
        You are a friendly assistant helping people with emails.
        When you decide to perform an action, explain your decision and then
        perform the action.
    </message>
"""
```

Plug-ins

Use [plug-ins](#) para fazer coisas que um LLM não pode fazer sozinho, como recuperar dados de fontes de dados externas ou concluir tarefas no mundo real.

Por exemplo, um LLM não pode enviar um email, portanto, para adicionar essa função a um aplicativo de chat, você precisaria criar um plug-in. Para processar texto dos emails, você pode usar [plug-ins principais](#), como o [ConversationSummaryPlugin](#).

Certifique-se de documentar claramente as funções no seus plug-ins, os planners usam essas informações para determinar quais funções estão disponíveis.

Planners

Um [planner](#) pode analisar as funções disponíveis e encontrar maneiras alternativas de atingir a meta.

Chamar as funções do plug-in nem sempre é eficiente. Por exemplo, digamos que você queira somar os números entre 1 e 100. Você poderia chamar um plug-in de matemática, mas o LLM precisaria fazer uma chamada separada para cada número.

Além disso, a melhor sequência e combinação de funções para atingir um objetivo depende dos detalhes. Por exemplo, suponha que você esteja criando um aplicativo auxiliar de chat por email e inclua um plug-in para permitir o envio de emails. No entanto, alguns emails podem precisar de uma ação diferente, como uma solicitação de reunião sem confirmação de presença, enviar uma resposta não é necessário, mas adicionar um item de calendário é. Um planner examina todas as funções disponíveis e apresenta maneiras eficientes de atingir as metas.

Copilotos adicionam interação com o usuário

A automação de processos tem muitos benefícios, mas às vezes o usuário precisa tomar decisões ao longo do processo. Um agente não pode automatizar as ações do usuário. É aí que entram os copilotos.

Um agente no seu aplicativo de chat por email pode produzir o seguinte plano para enviar um email:

1. Obter o endereço de email e o nome do usuário
2. Obter o endereço de email do destinatário
3. Obter o tópico do email
4. Gerar o assunto e o corpo do email
5. Enviar o email

Muito prático, mas e se o usuário não gostar do corpo do email? Um copilot adiciona uma etapa de interação do usuário ao plano:

1. Obter o endereço de email e o nome do usuário
2. Obter o endereço de email do destinatário
3. Obter o tópico do email
4. Gerar o assunto e o corpo do email
5. **Revisar o email com o usuário e fazer ajustes**
6. Enviar o email

Aplicativo Semantic Kernel Chat Copilot

Para começar a usar os copilotos, experimente o [Semantic Kernel Chat Copilot](#), um aplicativo de referência para criar uma experiência de chat com um agente de IA.

Conteúdo relacionado

- Desenvolver agentes de IA usando o OpenAI do Azure e o SDK do Semantic Kernel

Noções básicas de tokens

Artigo • 27/05/2024

Tokens são palavras, conjuntos de caracteres ou combinações de palavras e pontuação usadas por LLMs (modelos de linguagem grandes) para decompor o texto. A geração de tokens é a primeira etapa no treinamento. O LLM analisa as relações semânticas entre tokens, como a frequência com que eles são usados juntos ou se são usados em contextos semelhantes. Após o treinamento, o LLM usa esses padrões e relações para gerar uma sequência de tokens de saída com base na sequência de entrada.

Transformando texto em tokens

O conjunto de tokens exclusivos em que um LLM é treinado é conhecido como seu *vocabulário*.

Por exemplo, considere a seguinte sentença:

Eu ouvi um cachorro latir alto para um gato

Esse texto pode ter o token gerado como:

- I
- Ouvi
- um
- cachorro
- latido
- alto
- at
- um
- cat

Ao ter um conjunto suficientemente grande de texto de treinamento, a geração de tokens pode compilar um vocabulário de milhares de tokens.

Métodos comuns de geração de tokens

O método de geração de tokens específico varia de acordo com a LLM. Os métodos comuns de geração de tokens incluem:

- Geração de tokens de **palavra** (o texto é dividido em palavras individuais com base em um delimitador)
- Geração de tokens de **caracteres** (o texto é dividido em caracteres individuais)
- Geração de tokens de **subpalavra** (o texto é dividido em palavras parciais ou conjuntos de caracteres)

Por exemplo, os modelos GPT, desenvolvidos pela OpenAI, usam um tipo de geração de tokens de subpalavra que é conhecido como BPE (*Codificação de Par de Bytes*). O OpenAI fornece [uma ferramenta para visualizar como os tokens do texto serão gerados](#).

Há vantagens e desvantagens em cada método de geração de tokens:

[] Expandir a tabela

Tamanho do token	Vantagens	Desvantagens
Tokens menores (geração de tokens de subpalavra ou caractere)	<ul style="list-style-type: none"> - Permite que o modelo manipule um intervalo maior de entradas, como palavras desconhecidas, erros de digitação ou sintaxe complexa. - Pode permitir que o tamanho do vocabulário seja reduzido, exigindo menos recursos de memória. 	<ul style="list-style-type: none"> - Um determinado texto é dividido em mais tokens, exigindo recursos computacionais adicionais durante o processamento - Dado um limite de token fixo, o tamanho máximo da entrada e da saída do modelo é menor
Tokens maiores (geração de tokens de palavra)	<ul style="list-style-type: none"> - Um determinado texto é dividido em menos tokens, exigindo menos recursos computacionais durante o processamento. - Dado o mesmo limite de token, o tamanho máximo da entrada e da saída do modelo é maior. 	<ul style="list-style-type: none"> - Pode causar um vocabulário maior, exigindo mais recursos de memória. - Pode limitar a capacidade dos modelos de lidar com palavras desconhecidas, erros de digitação ou sintaxe complexa.

Como os LLMs usam tokens

Depois que a LLM conclui a geração de tokens, ela atribui uma ID a cada token exclusivo.

Considere nossa frase de exemplo:

Eu ouvi um cachorro latir alto para um gato

Depois que o modelo usa um método de geração de tokens de palavra, ele pode atribuir IDs de token da seguinte maneira:

- Eu (1)
- ouvi (2)
- um (3)
- cachorro (4)
- latir (5)
- alto (6)
- para (7)
- a (o token "a" já está atribuído a uma ID de 3)
- gato (8)

Ao atribuir IDs, o texto pode ser representado como uma sequência de IDs de token. A frase de exemplo seria representada como [1, 2, 3, 4, 5, 6, 7, 3, 8]. A frase "Ouvi um gato" seria representada como [1, 2, 3, 8].

À medida que o treinamento continua, o modelo adiciona novos tokens no texto de treinamento ao seu vocabulário e atribui a ele uma ID. Por exemplo:

- miau (9)
- executar (10)

As relações semânticas entre os tokens podem ser analisadas usando essas sequências de ID de token. Vetores numéricos de vários valores, conhecidos como [inserções](#), são usados para representar essas relações. Uma inserção é atribuída a cada token com base na frequência com que ele é usado junto ou em contextos semelhantes aos outros tokens.

Depois de treinado, um modelo pode calcular uma inserção para um texto que contém vários tokens. O modelo gera tokens do texto e calcula um valor geral de inserções com base nas inserções aprendidas dos tokens individuais. Essa técnica pode ser usada para pesquisas semânticas de documentos ou para adicionar [memórias](#) a uma IA.

Durante a geração de saída, o modelo prevê um valor de vetor para o próximo token na sequência. Em seguida, o modelo seleciona o próximo token em seu vocabulário com base nesse valor de vetor. Na prática, o modelo calcula vários vetores usando vários elementos das inserções dos tokens anteriores. Em seguida, o modelo avalia todos os tokens potenciais desses vetores e seleciona o mais provável para continuar a sequência.

A geração de saída é uma operação iterativa. O modelo acrescenta o token previsto à sequência até agora e o usa como entrada para a próxima iteração, criando a saída final de um token de cada vez.

Limites de token

Os LLMs têm limitações em relação ao número máximo de tokens que podem ser usados como entrada ou gerados como saída. Essa limitação geralmente faz com que os tokens de entrada e saída sejam combinados em uma janela de contexto máxima.

Por exemplo, o GPT-4 dá suporte a até 8.192 tokens de contexto. O tamanho combinado dos tokens de entrada e saída não pode exceder 8.192.

Juntos, o limite de token e o método de geração de tokens de um modelo determinam o comprimento máximo do texto que pode ser fornecido como entrada ou gerado como saída.

Por exemplo, considere um modelo que tenha uma janela de contexto máxima de 100 tokens. O modelo processa nossas frases de exemplo como texto de entrada:

Eu ouvi um cachorro latir alto para um gato

Usando um método de geração de tokens baseado em palavras, a entrada é de nove tokens. Isso deixa 91 tokens de **palavra** disponíveis para a saída.

Usando um método de geração de tokens baseado em caracteres, a entrada é de 34 tokens (incluindo espaços). Isso deixa somente 66 tokens de **caractere** disponíveis para a saída.

Preço baseado em token e limitação de taxa

Os serviços de IA generativos geralmente usam preços baseados em token. O custo de cada solicitação depende do número de tokens de entrada e saída. O preço pode ser diferente entre entrada e saída. Por exemplo, consulte os [Preços do Serviço OpenAI do Azure](#).

Os serviços de IA generativos também podem ser limitados em relação ao número máximo de tokens por minuto (TPM). Esses limites de taxa podem variar dependendo da região do serviço e da LLM. Para obter mais informações sobre regiões específicas, consulte [Cotas e limites do Serviço OpenAI do Azure](#).

Conteúdo relacionado

- [Como a IA e os LLMs generativos funcionam](#)
- [Noções básicas sobre inserções](#)
- [Trabalhando com bancos de dados vetoriais](#)

 Colaborar conosco no
GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).

.NET

Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 Abrir um problema de documentação

 Fornecer comentários sobre o produto

Inserções no .NET

Artigo • 27/05/2024

As inserções são a maneira como os LLMs capturam o significado semântico. São representações numéricas de dados não numéricos que um LLM pode usar para determinar relações entre conceitos. Você pode usar inserções para ajudar um modelo de IA a entender o significado das entradas para que ele possa executar comparações e transformações, como resumir texto ou criar imagens de descrições de texto. As LLMs podem usar inserções imediatamente e você pode armazenar inserções em bancos de dados vetoriais para fornecer memória semântica para LLMs conforme necessário.

Casos de uso para inserções

Esta seção lista os principais casos de uso para inserções.

Use seus próprios dados para melhorar a relevância da conclusão

Use seus próprios bancos de dados para gerar inserções para seus dados e integrá-los a uma LLM para disponibilizá-los para conclusões. Esse uso de inserções é um componente importante da [geração aumentada de recuperação](#).

Aumentar a quantidade de texto que você pode ajustar em um prompt

Use inserções para aumentar a quantidade de contexto que você pode ajustar em um prompt sem aumentar o número de tokens necessários.

Por exemplo, suponha que você queira incluir 500 páginas de texto em um prompt. O número de tokens para esse texto bruto excederá o limite do token de entrada, impossibilitando a inclusão direta em um prompt. Você pode usar inserções para resumir e dividir grandes quantidades desse texto em partes que são pequenas o suficiente para caber em uma entrada e avaliar a similaridade de cada peça com todo o texto bruto. Em seguida, você pode escolher uma peça que preserve melhor o significado semântico do texto bruto e usá-la em seu prompt sem atingir o limite de token.

Executar classificação de texto, resumo ou tradução

Use inserções para ajudar um modelo a entender o significado e o contexto do texto e, em seguida, classificar, resumir ou traduzir esse texto. Por exemplo, você pode usar inserções para ajudar modelos a classificar textos como positivos ou negativos, spam ou não spam, ou notícias ou opiniões.

Gerar e transcrever áudio

Use inserções de áudio para processar arquivos de áudio ou entradas em seu aplicativo.

Por exemplo, o [Serviço de Fala](#) dá suporte a diversas inserções de áudio, incluindo [conversão de fala em texto](#) e [conversão de texto em fala](#). Você pode processar áudio em tempo real ou em lotes.

Transformar texto em imagens ou imagens em texto

O processamento semântico de imagem requer inserções de imagem, que a maioria das LLMs não pode gerar. Use um modelo de inserção de imagem, como [ViT](#) para criar inserções de vetor para imagens. Em seguida, você pode usar essas inserções com um modelo de geração de imagem para criar ou modificar imagens usando texto ou vice-versa. Por exemplo, você pode [usar o modelo DALL·E para gerar imagens](#) como logotipos, rostos, animais e paisagens.

Gerar ou documentar código

Use inserções para ajudar um modelo a criar código a partir de texto ou vice-versa, convertendo diferentes expressões de código ou texto em uma representação comum. Por exemplo, você pode usar inserções para ajudar um modelo a gerar ou documentar código em C# ou Python.

Escolher um modelo de inserção

Você gera inserções para seus dados brutos usando um modelo de inserção de IA, que pode codificar dados não numéricos em um vetor (uma longa matriz de números). O modelo também pode decodificar uma inserção em dados não numéricos que têm o mesmo significado ou um significado semelhante aos dados brutos originais. Há muitos modelos de inserção disponíveis para uso, com o modelo `text-embedding-ada-002` do OpenAI sendo um dos modelos comuns usados. Para obter mais exemplos, consulte a lista de [Modelos de inserção disponíveis no Azure OpenAI](#).

Armazenar e processar inserções em um banco de dados de vetor

Depois de gerar inserções, você precisará de uma maneira de armazená-las para que possa recuperá-las posteriormente com chamadas para uma LLM. Os bancos de dados vetoriais são projetados para armazenar e processar vetores, portanto, eles são uma casa natural para inserções. Diferentes bancos de dados vetoriais oferecem diferentes recursos de processamento, portanto, você deve escolher um com base em seus dados brutos e suas metas. Para obter informações sobre suas opções, consulte [soluções de banco de dados de vetor disponíveis](#).

Usando inserções em sua solução LLM

Ao criar aplicativos baseados em LLM, você pode usar o Kernel Semântico para integrar modelos de inserção e repositórios de vetores, para que você possa efetuar pull rápido de dados de texto e gerar e armazenar inserções. Isso permite que você use uma solução de banco de dados de vetor para armazenar e recuperar memórias semânticas.

Conteúdo relacionado

- [Como funcionam o GenAI e as LLMs](#)
- [Geração aumentada por recuperação](#)
- [Treinamento: desenvolver agentes de IA com o Azure OpenAI e o Kernel Semântico](#)

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).

.NET

Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 [Abrir um problema de documentação](#)

 [Fornecer comentários sobre o produto](#)

Bancos de dados vetoriais para .NET + IA

Artigo • 24/11/2024

Os bancos de dados vetoriais são projetados para armazenar e gerenciar [inserções](#) de vetores. Inserções são representações numéricas de dados não numéricos que preservam o significado semântico. Palavras, documentos, imagens, áudio e outros tipos de dados podem ser vetorizados. Você pode usar inserções para ajudar um modelo de IA a entender o significado das entradas para que ele possa realizar comparações e transformações, como resumir textos, encontrar dados contextualmente relacionados ou criar imagens a partir de descrições de texto.

Por exemplo, você pode usar um banco de dados vetorial para:

- Identificar imagens, documentos e músicas semelhantes com base no seus conteúdos, temas, sentimentos e estilos.
- Identificar produtos semelhantes com base em suas características, recursos e grupos de usuários.
- Recomendar conteúdo, produtos ou serviços com base nas preferências do usuário.
- Identificar as melhores opções potenciais em um grande conjunto de escolhas para atender a requisitos complexos.
- Identificar anomalias de dados ou atividades fraudulentas que sejam diferentes dos padrões predominantes ou normais.

Entenda a busca em vetores

Os bancos de dados vetoriais oferecem recursos de busca em vetores para encontrar itens semelhantes com base em suas características de dados, em vez de correspondências exatas em um campo de propriedade. A busca em vetores funciona analisando as representações vetoriais de seus dados que você criou usando um modelo de inserção de IA, como os modelos de inserção do [OpenAI do Azure](#). O processo de pesquisa mede a distância entre os vetores de dados e seu vetor de consulta. Os vetores de dados mais próximos do vetor de consulta são os mais semelhantes semanticamente.

Alguns serviços, como o [Azure Cosmos DB for MongoDB vCore](#), fornecem recursos nativos de busca em vetores para seus dados. Outros bancos de dados podem ser aprimorados com a busca em vetores indexando os dados armazenados usando um serviço como a Pesquisa de IA do Azure, que pode verificar e indexar seus dados para fornecer recursos de busca em vetores.

Fluxos de trabalho de busca em vetores com .NET e OpenAI

Os bancos de dados vetoriais e seus recursos de pesquisa são especialmente úteis em fluxos de trabalho do padrão [RAG](#) com o OpenAI do Azure. Esse padrão permite que você aumente ou aprimore seu modelo de IA com conhecimento adicional semanticamente avançado de seus dados. Um fluxo de trabalho de IA comum usando bancos de dados vetoriais pode incluir as seguintes etapas:

1. Criar inserções para seus dados usando um modelo de inserção do OpenAI.
2. Armazenar e indexar as inserções em um banco de dados vetorial ou serviço de pesquisa.
3. Converter os prompts de usuário do seu aplicativo em inserções.
4. Executar uma busca em vetores nos seus dados, comparando a inserção do prompt do usuário com as inserções do seu banco de dados.
5. Usar um modelo de linguagem, como GPT-35 ou GPT-4, para montar uma conclusão amigável para o usuário a partir dos resultados da busca em vetores.

Visite o tutorial [Implementar o OpenAI do Azure com RAG usando a busca em vetores em um aplicativo .NET](#) para obter um exemplo prático desse fluxo.

Outros benefícios do padrão RAG incluem:

- Gerar respostas contextualmente relevantes e precisas para prompts de usuários a partir de modelos de IA.
- Superar os limites de tokens do LLM: o trabalho pesado é feito por meio da busca em vetores do banco de dados.
- Reduzir os custos do ajuste fino frequente em dados atualizados.

Soluções de bancos de dados vetoriais disponíveis

Os aplicativos de IA geralmente usam bancos de dados e serviços de vetor de dados para melhorar a relevância e fornecer funcionalidade personalizada. Muitos desses serviços fornecem um SDK nativo para .NET, enquanto outros oferecem um serviço REST ao qual você pode se conectar por meio de um código personalizado. O Kernel Semântico fornece um modelo de componente extensível que permite que você use repositórios de vetores diferentes sem a necessidade de aprender cada SDK.

O Kernel Semântico fornece conectores para os seguintes bancos de dados e serviços de vetor:

 Expandir a tabela

Serviço vetorial	Conector do Kernel Semântico	SDK .NET
Azure AI Search	Microsoft.SemanticKernel.Connectors.AzureAISearch	Azure.Search.Documents
Azure Cosmos DB para NoSQL	Microsoft.SemanticKernel.Connectors.AzureCosmosDBNoSQL	Microsoft.Azure.Cosmos
Azure Cosmos DB for MongoDB	Microsoft.SemanticKernel.Connectors.AzureCosmosDBMongoDB	MongoDb.Driver
Servidor PostgreSQL do Azure	Microsoft.SemanticKernel.Connectors.Postgres	Npgsql
Banco de Dados SQL do Azure	Microsoft.SemanticKernel.Connectors.SqlServer	Microsoft.Data.SqlClient
Chroma	Microsoft.SemanticKernel.Connectors.Chroma	NA
DuckDB	Microsoft.SemanticKernel.Connectors.DuckDB	DuckDB.NET.Data.Full
Milvus	Microsoft.SemanticKernel.Connectors.Milvus	Milvus.Client
Busca em vetores do MongoDB Atlas	Microsoft.SemanticKernel.Connectors.MongoDB	MongoDb.Driver
Pinecone	Microsoft.SemanticKernel.Connectors.Pinecone	REST API
Postgres	Microsoft.SemanticKernel.Connectors.Postgres	Npgsql
Qdrant	Microsoft.SemanticKernel.Connectors.Qdrant	Qdrant.Client
Redis	Microsoft.SemanticKernel.Connectors.Redis	StackExchange.Redis
Weaviate	Microsoft.SemanticKernel.Connectors.Weaviate	REST API

Visite a documentação de cada serviço respectivo para descobrir o SDK do .NET e o suporte à API.

Conteúdo relacionado

- [Implementar o OpenAI do Azure com o RAG usando a busca em vetores em um aplicativo .NET](#)

- Mais conectores .NET do Semantic Kernel ↗

Engenharia de solicitação no .NET

Artigo • 27/06/2024

Neste artigo, você vai explorar os conceitos essenciais de engenharia de prompt. Muitos modelos de IA são baseados em prompts, o que significa que respondem ao texto de entrada do usuário (um *prompt*) com uma resposta gerada por algoritmos preditivos (um *preenchimento*). Modelos mais recentes também costumam ser compatíveis com preenchimentos em formato de chat com mensagens baseadas em funções (sistema, usuário, assistente) e um histórico de chat para preservar as conversas.

Trabalhe com prompts

Considere este exemplo de geração de texto em que a *solicitação* é a entrada do usuário e o *preenchimento* é a saída do modelo:

Solicitação: "O presidente que teve um mandato curto foi"

Preenchimento: "*Pedro Lascurain*".

O preenchimento parece correto, mas o que dizer se o seu aplicativo se destina a ajudar os estudantes de história dos EUA? O mandato de 45 minutos de Pedro Lascurain foi o mandato mais curto de que já se teve notícia, mas ele foi presidente do México. Os estudantes de história dos EUA provavelmente estão procurando por "*William Henry Harrison*". Claramente, o aplicativo pode ser mais útil para seus usuários pretendidos se você forneceu contexto a ele.

A engenharia de prompt adiciona contexto ao prompt fornecendo *instruções*, *exemplos* e *pistas* para ajudar o modelo a produzir preenchimentos melhores.

Os modelos GPT compatíveis com geração de texto não requerem nenhum formato específico, mas você deve organizar seus prompts de modo que fique claro o que é uma instrução e o que é um exemplo. Os modelos GPT compatíveis com aplicativos baseados em chat usam três funções para organizar preenchimentos: uma função de sistema que controla o chat, uma função de usuário para representar a entrada do usuário e uma função de assistente para responder aos usuários. Você divide seus prompts em mensagens para cada função:

- *Mensagens do sistema* dão instruções sobre o assistente ao modelo. Uma solicitação pode ter apenas uma mensagem do sistema e deve ser a primeira mensagem.

- As *mensagens do usuário* incluem prompts do usuário e mostram exemplos e prompts históricos, ou contêm instruções para o assistente. Um exemplo de preenchimento de chat deve ter pelo menos uma mensagem de usuário.
- As *mensagens do assistente* mostram exemplos ou históricos de preenchimentos e devem conter uma resposta à mensagem anterior do usuário. Mensagens de assistente não são necessárias, mas se você incluir uma, elas deverão ser emparelhadas com uma mensagem de usuário para formar um exemplo.

Use instruções para aprimorar o preenchimento

Uma instrução é um texto que informa ao modelo como responder. Uma instrução pode ser uma diretiva ou um imperativo:

- *Diretivas* dizem ao modelo como se comportar, mas não são comandos simples. Pense na preparação do personagem para um ator de improviso: "**Você está ajudando os alunos a aprender sobre a história dos EUA, então fale sobre os EUA, a menos que perguntem especificamente sobre outros países**"
- *Imperativos* são comandos claros para o modelo seguir. "**Traduzir para tagalo:**"

Diretivas são mais abertas e flexíveis do que imperativos:

- Você pode combinar várias diretivas em uma instrução.
- As instruções geralmente funcionam melhor quando você as usa com exemplos. No entanto, como os imperativos são comandos claros, os modelos não precisam de exemplos para entendê-los (embora você possa usar um exemplo para mostrar ao modelo como formatar respostas). Como uma diretiva não informa ao modelo exatamente o que fazer, cada exemplo pode ajudar o modelo a funcionar melhor.
- Normalmente, é melhor dividir uma instrução difícil em uma série de etapas, o que você pode fazer com uma sequência de diretivas. Você também deve informar o modelo para gerar o resultado de cada etapa, para poder fazer ajustes granulares facilmente. Embora você possa dividir a instrução em etapas por conta própria, é mais fácil apenas dizer ao modelo para fazê-lo e gerar o resultado de cada etapa. Essa abordagem é chamada [solicitação de cadeia de pensamento](#).

Contexto de adição de conteúdo principal e de apoio

Você pode fornecer conteúdo para adicionar mais contexto às instruções.

Conteúdo principal é o texto que você deseja que o modelo processe com uma instrução. Seja qual for a ação que a instrução implicar, o modelo a executará no

conteúdo principal para produzir um preenchimento.

Conteúdo de suporte é um texto que você faz referência em uma instrução, mas que não é o alvo da instrução. O modelo usa o conteúdo de suporte para concluir a instrução, o que significa que o suporte ao conteúdo também aparece em preenchimentos, normalmente como algum tipo de estrutura (como em rótulos de coluna ou títulos).

Use rótulos com seu conteúdo instrucional para ajudar o modelo a descobrir como usá-lo com a instrução. Não se preocupe muito com a precisão. Os rótulos não precisam corresponder exatamente às instruções porque o modelo lidará com coisas como capitalização e formato de palavra.

Suponha que você use a instrução "Resumir realizações presidenciais dos EUA" para produzir uma lista. O modelo organizá-la e ordená-la de várias maneiras. Mas e se você quiser que a lista agrupe as realizações por um conjunto específico de categorias? Use o conteúdo de suporte para adicionar essas informações à instrução.

Ajuste a instrução para que o modelo agrupe por categoria e acrescente o conteúdo de suporte que especifica essas categorias:

```
C#  
  
prompt = """  
Instructions: Summarize US Presidential accomplishments, grouped by  
category.  
Categories: Domestic Policy, US Economy, Foreign Affairs, Space Exploration.  
Accomplishments: 'George Washington  
- First president of the United States.  
- First president to have been a military veteran.  
- First president to be elected to a second term in office.  
- Received votes from every presidential elector in an election.  
- Filled the entire body of the United States federal judges; including the  
Supreme Court.  
- First president to be declared an honorary citizen of a foreign country,  
and an honorary citizen of France.  
John Adams ...' //Text truncated  
""";  
""";
```

Use exemplos para orientar o modelo

Um exemplo é o texto que mostra ao modelo como responder fornecendo entrada de usuário e saída de modelo de exemplo. O modelo usa exemplos para inferir o que incluir nos preenchimentos. Exemplos podem vir antes ou depois das instruções em uma solicitação construída, mas as duas não devem ser intercaladas.

Um exemplo começa com um prompt e pode, opcionalmente, incluir um preenchimento. Um preenchimento em um exemplo não precisa incluir a resposta literal. Ele pode conter apenas uma palavra formatada, o primeiro marcador em uma lista não ordenada ou algo semelhante para indicar como cada preenchimento deve ser iniciado.

Os exemplos são classificados como [Aprendizado sem nenhuma tentativa](#) ou [aprendizado de poucas tentativas](#) com base em se contêm preenchimentos literais.

- Exemplos de **aprendizado sem nenhuma tentativa** incluem uma solicitação sem preenchimento literal. Essa abordagem testa as respostas de um modelo sem lhe dar um exemplo de saída de dados. Os prompts zero-shot (sem treinamento anterior) podem ter preenchimentos que incluem pistas, como indicar que o modelo deve gerar uma lista ordenada ao incluir "1." como o preenchimento.
- Exemplos de **aprendizado com poucas tentativas** incluem vários pares de solicitações com preenchimentos literais. O aprendizado com poucas tentativas pode alterar o comportamento do modelo ao adicionar ao seu conhecimento existente.

Entenda as pistas

Uma sugestão é um texto que transmite a estrutura ou o formato de saída desejado. Como uma instrução, uma sugestão não é processada pelo modelo como se fosse entrada de usuário. Como um exemplo, uma sugestão mostra ao modelo o que você deseja em vez de dizer o que fazer. Você pode adicionar quantas sugestões desejar, para poder iterar para obter o resultado desejado. As sugestões são usadas com uma instrução ou um exemplo e devem estar no final da solicitação.

Suponha que você use uma instrução para informar ao modelo para produzir uma lista de realizações presidenciais por categoria, juntamente com o conteúdo de suporte que informa ao modelo quais categorias usar. Você decide que o modelo deve produzir uma lista aninhada em maiúsculo para categorias, com as realizações de cada presidente em cada categoria listada em uma linha que começa com o respectivo nome, com presidentes listados cronologicamente. Após a instrução e o conteúdo de suporte, você pode adicionar três sugestões para mostrar ao modelo como estruturar e formatar a lista:

```
C#
```

```
prompt = """
Instructions: Summarize US Presidential accomplishments, grouped by
category.
Categories: Domestic Policy, US Economy, Foreign Affairs, Space Exploration.
```

Accomplishments: George Washington
First president of the United States.
First president to have been a military veteran.
First president to be elected to a second term in office.
First president to receive votes from every presidential elector in an election.
First president to fill the entire body of the United States federal judges; including the Supreme Court.
First president to be declared an honorary citizen of a foreign country, and an honorary citizen of France.
John Adams ... /// Text truncated

DOMESTIC POLICY

- George Washington:
 - John Adams:
- """;

- **POLÍTICA DOMÉSTICA** mostra ao modelo que você deseja que ele inicie cada grupo com a categoria em maiúsculo.
- – **George Washington**: mostra ao modelo para começar cada seção com as realizações de George Washington listadas em uma linha.
- – **John Adams**: mostra ao modelo que deve listar os presidentes restantes em ordem cronológica.

Exemplo de prompt usando o .NET

O .NET fornece várias ferramentas para criar prompts e conversar com diferentes modelos de IA. Use o [Kernel Semântico](#) para se conectar a uma ampla variedade de modelos e serviços de IA e também a outros SDKs, como a [biblioteca do .NET da OpenAI](#) oficial. O Kernel Semântico inclui ferramentas para criar prompts com diferentes funções e manter o histórico de chat, além de muitos outros recursos.

Considere o seguinte exemplo de código:

C#

```
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.ChatCompletion;

// Create a kernel with OpenAI chat completion
#pragma warning disable SKEXP0010
Kernel kernel = Kernel.CreateBuilder()
    .AddOpenAIChatCompletion(
        modelId: "phi3:mini",
        endpoint: new Uri("http://localhost:11434"),
        apiKey: "")  
.Build();
```

```

var aiChatService = kernel.GetRequiredService<IChatCompletionService>();
var chatHistory = new ChatHistory();
chatHistory.Add(
    new ChatMessageContent(AuthorRole.System, "You are a helpful AI
Assistant."));

while (true)
{
    // Get user prompt and add to chat history
    Console.WriteLine("Your prompt:");
    chatHistory.Add(new ChatMessageContent(AuthorRole.User,
Console.ReadLine()));

    // Stream the AI response and add to chat history
    Console.WriteLine("AI Response:");
    var response = "";
    await foreach (var item in
        aiChatService.GetStreamingChatMessageContentsAsync(chatHistory))
    {
        Console.Write(item.Content);
        response += item.Content;
    }
    chatHistory.Add(new ChatMessageContent(AuthorRole.Assistant, response));
    Console.WriteLine();
}

```

O código anterior fornece exemplos dos seguintes conceitos:

- Cria um serviço de histórico de chat para solicitar preenchimentos ao modelo de IA com base em funções de autor.
- Configura a IA com uma mensagem de `AuthorRole.System`.
- Aceita a entrada do usuário para permitir diferentes tipos de prompts no contexto de um `AuthorRole.User`.
- Transmite o preenchimento de IA de forma assíncrona para fornecer uma experiência de chat dinâmica.

Amplie suas técnicas de engenharia de prompt

Você também pode aumentar o poder de seus prompts com técnicas de engenharia de prompt mais avançadas que são abordadas em profundidade em seus próprios artigos.

- Os LLMs têm limites de entrada de token que restringem a quantidade de texto que você pode ajustar em uma solicitação. Use [incorporações](#) e [soluções de banco de dados vetorial](#) para reduzir o número de tokens necessários para representar um determinado trecho de texto.
- Os LLMs não são treinados em seus dados, a menos que você os treine por conta própria, o que pode ser caro e demorado. Use a [geração aumentada de](#)

recuperação (RAG) para disponibilizar seus dados para um LLM sem treiná-lo.

Conteúdo relacionado

- [Técnicas de engenharia de prompt](#)
- [Configurar solicitações no Kernel Semântico](#)

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).

.NET

Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 [Abrir um problema de documentação](#)

 [Fornecer comentários sobre o produto](#)

Solicitação de cadeia de pensamento

Artigo • 27/05/2024

Este artigo explica o uso da cadeia de solicitação de reflexões no .NET.

O desempenho do modelo GPT aproveita a *engenharia de prompts*, que é a prática de fornecer instruções e os exemplos de um modelo para preparar ou refinar a saída. À medida que processam instruções, os modelos cometem mais erros de raciocínio quando tentam responder imediatamente, em vez de dar um tempo para descobrir uma resposta. Você pode ajudar o modelo a explicar seu caminho para respostas corretas de forma mais confiável solicitando que o modelo inclua sua cadeia de reflexões, ou seja, as etapas executadas para seguir uma instrução, juntamente com os resultados de cada etapa.

A *solicitação de cadeia de reflexões* é a prática de solicitar que um modelo GPT execute uma tarefa passo a passo e apresente cada etapa e seu resultado em ordem na saída. Isso simplifica a engenharia de prompt descarregando parte do planejamento de execução para o modelo e facilita a conexão de qualquer problema com uma etapa específica para que você saiba onde concentrar mais esforços.

Geralmente, é mais simples apenas instruir o modelo a incluir sua cadeia de reflexões, mas você pode usar exemplos para mostrar ao modelo como dividir tarefas. As seções a seguir mostram as duas maneiras.

Usar a cadeia de solicitação de reflexões em instruções

Para usar uma instrução para a cadeia de solicitação de reflexões, inclua uma diretiva que informe ao modelo para executar a tarefa passo a passo e gerar o resultado de cada etapa.

C#

```
prompt= "Instructions: Compare the pros and cons of EVs and petroleum-fueled vehicles. Break the task into steps, and output the result of each step as you perform it.;"
```

Usar a cadeia de solicitação de reflexões em exemplos

Você pode usar exemplos para indicar as etapas para a cadeia de solicitação de reflexões, que o modelo interpretará que também deve gerar os resultados da etapa. As etapas podem incluir sugestões de formatação.

```
C#  
  
prompt= """  
    Instructions: Compare the pros and cons of EVs and petroleum-fueled  
    vehicles.  
  
    Differences between EVs and petroleum-fueled vehicles:  
    -  
  
    Differences ordered according to overall impact, highest-impact  
    first:  
    1.  
  
    Summary of vehicle type differences as pros and cons:  
    Pros of EVs  
    1.  
    Pros of petroleum-fueled vehicles  
    1.  
    """;
```

Conteúdo relacionado

- Técnicas de engenharia de prompt

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).



Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 [Abrir um problema de documentação](#)

 [Fornecer comentários sobre o produto](#)

Aprendizado zero-shot e few-shot

Artigo • 27/05/2024

Este artigo explica o aprendizado zero-shot e o aprendizado few-shot para engenharia de prompt no .NET, incluindo os principais casos de uso.

O desempenho do modelo GPT aproveita a *engenharia de prompts*, a prática de fornecer instruções e os exemplos de um modelo para refinar a saída. O aprendizado zero-shot e o aprendizado few-shot são técnicas que você pode usar ao fornecer exemplos.

Com o aprendizado zero-shot, você inclui prompts, mas não conclusões verbatim. Você pode incluir conclusões que consistem apenas em indicações. O aprendizado zero-shot depende inteiramente do conhecimento existente do modelo para gerar respostas, o que reduz o número de tokens criados e pode ajudar você a controlar os custos. No entanto, o aprendizado zero-shot não adiciona ao conhecimento do modelo.

Este é um exemplo de prompt zero-shot que informa ao modelo para avaliar a entrada do usuário para determinar quais das quatro intenções possíveis a entrada representa e, em seguida, para preceder a resposta com "Intent: ".

C#

```
prompt = """  
Instructions: What is the intent of this request?  
If you don't know the intent, don't guess; instead respond with "Unknown".  
Choices: SendEmail, SendMessage, CompleteTask, CreateDocument, Unknown.  
User Input: {request}  
Intent:  
""";
```

Com o aprendizado few-shot, você inclui prompts emparelhados com conclusões verbatim. Em comparação com o aprendizado zero-shot, isso significa que o aprendizado few-shot produz mais tokens e faz com que o modelo atualize o conhecimento, o que pode fazer com que o aprendizado few-shot tenha mais recursos. No entanto, pelos mesmos motivos, o aprendizado few-shot também ajuda o modelo a produzir respostas mais relevantes.

C#

```
prompt = """  
Instructions: What is the intent of this request?  
If you don't know the intent, don't guess; instead respond with "Unknown".  
Choices: SendEmail, SendMessage, CompleteTask, CreateDocument, Unknown.
```

User Input: Can you send a very quick approval to the marketing team?

Intent: SendMessage

User Input: Can you send the full update to the marketing team?

Intent: SendEmail

User Input: {request}

Intent:

""";

Casos de uso de aprendizado zero-shot

O aprendizado zero-shot é a prática de passar prompts que não são emparelhados com conclusões verbatim, embora possam ser emparelhados com uma sugestão. Há dois casos de uso primários para aprendizado zero-shot:

- **Trabalhar com LLMs ajustados** – Como depende do conhecimento existente do modelo, o aprendizado zero-shot não tem tantos recursos quanto o aprendizado few-shot e funciona bem com LLMs que já foram ajustados em conjuntos de dados de instrução. Você pode contar apenas com o aprendizado zero-shot e manter os custos relativamente baixos.
- **Estabelecer linhas de base de desempenho** – O aprendizado zero-shot pode ajudar você a simular o desempenho do aplicativo para os usuários reais. Isso permite avaliar vários aspectos do desempenho atual do modelo, como exatidão ou precisão. Nesse caso, você normalmente usa o aprendizado zero-shot para estabelecer uma linha de base de desempenho e, em seguida, experimenta o aprendizado few-shot para melhorar o desempenho.

Casos de uso de aprendizado few-shot

O aprendizado few-shot é a prática de passar prompts emparelhados com conclusões verbatim (prompts few-shot) para mostrar ao modelo como responder. Ao contrário do aprendizado zero-shot, o aprendizado few-shot pode adicionar ao conhecimento do modelo. Você pode até mesmo usar seus próprios conjuntos de dados para gerar prompts few-shot automaticamente, executando a geração aumentada por recuperação.

O aprendizado few-shot tem dois casos de uso primários:

- **Ajustar um LLM** – Como pode adicionar ao conhecimento do modelo, o aprendizado few-shot pode melhorar o desempenho de um modelo. Isso também faz com que o modelo crie mais tokens do que o aprendizado zero-shot, o que

pode eventualmente se tornar proibitivamente caro ou até mesmo inviável. No entanto, se o LLM ainda não estiver ajustado, você não terá um bom desempenho com prompts zero-shot e o aprendizado few-shot será garantido.

- **Correção de problemas de desempenho** – Você pode usar o aprendizado few-shot como uma continuação do aprendizado zero-shot. Nesse caso, você usa o aprendizado zero-shot para estabelecer uma linha de base de desempenho e, em seguida, experimenta o aprendizado few-shot com base nos prompts zero-shot usados. Isso permite que você adicione ao conhecimento do modelo depois de ver como ele responde no momento, para que você possa iterar e melhorar o desempenho, minimizando o número de tokens apresentados.

Advertências

- O aprendizado baseado em exemplo não funciona bem para tarefas de raciocínio complexas. No entanto, adicionar instruções pode ajudar a resolver isso.
- O aprendizado few-shot requer a criação de prompts longos. Prompts com grande número de tokens podem aumentar a computação e a latência. Isso normalmente significa um aumento de custos. Há também um limite para o tamanho dos prompts.
- Quando você usa vários exemplos, o modelo pode aprender padrões falsos, como "Sentimentos têm duas vezes mais chances de serem positivos do que negativos".

Conteúdo relacionado

- [Técnicas de engenharia de prompt](#)
- [Como funcionam o GenAI e os LLMs](#)

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).

.NET

Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 [Abrir um problema de documentação](#)

 [Fornecer comentários sobre o produto](#)

A RAG (geração aumentada de recuperação) fornece conhecimento de LLM

Artigo • 24/11/2024

Este artigo descreve como a geração aumentada de recuperação permite que os LLMs tratem as fontes de dados como conhecimento sem a necessidade de treinamento.

Os LLMs têm extensas bases de dados de conhecimento por meio do treinamento. Para a maioria dos cenários, você pode selecionar um LLM criado de acordo com os seus requisitos, mas esses LLMs ainda exigirão treinamento adicional para entender os dados específicos. A geração aumentada de recuperação permite disponibilizar os dados para os LLMs sem treiná-los primeiro.

Como a RAG funciona

Para executar a geração aumentada de recuperação, você cria inserções para os dados, juntamente com perguntas comuns sobre eles. Você pode fazer isso em tempo real ou criar e armazenar as inserções usando uma solução de banco de dados vetorial.

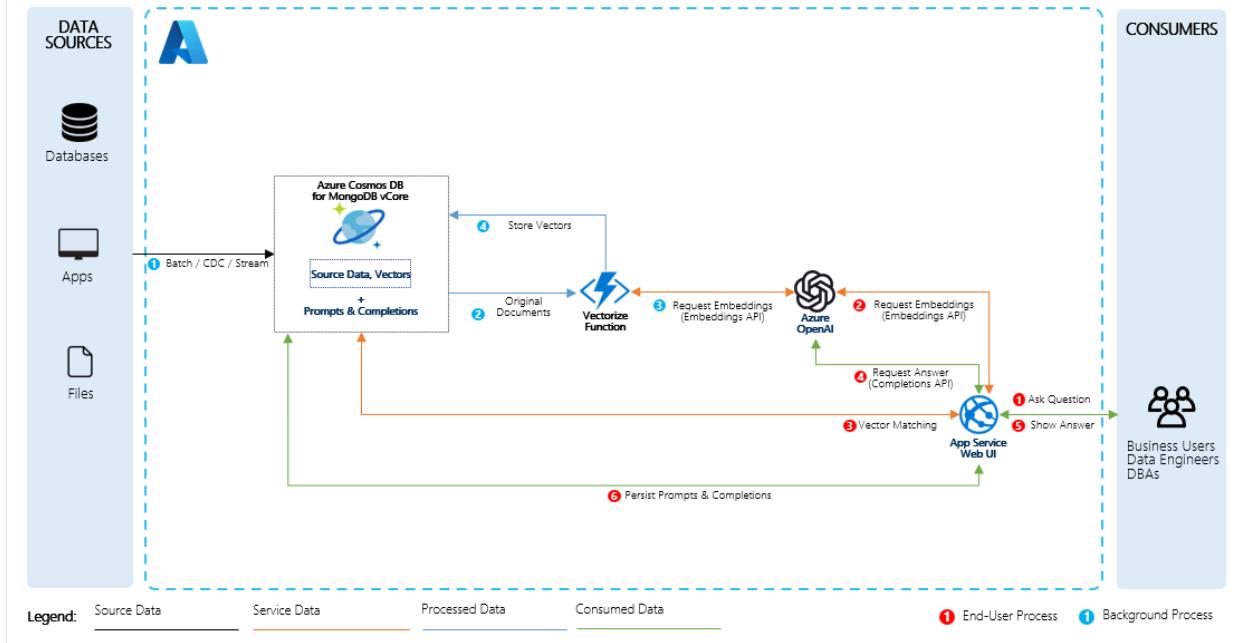
Quando um usuário faz uma pergunta, o LLM usa as inserções para comparar a pergunta do usuário com os dados e encontrar o contexto mais relevante. Esse contexto e a pergunta do usuário, em seguida, vão para o LLM em um prompt e o LLM fornece uma resposta com base nos dados.

Processo básico de RAG

Para executar a RAG, você deve processar cada fonte de dados que deseja usar para recuperações. O processo básico é como descrito a seguir:

1. Agrupe os dados grandes em partes gerenciáveis.
2. Converta as partes em um formato pesquisável.
3. Armazene os dados convertidos em um local que permita acesso eficiente. Além disso, é importante armazenar metadados relevantes para citações ou referências quando o LLM fornece respostas.
4. Alimente os dados convertidos nos LLMs em prompts.

Vector Search & AI Assistant for Azure Cosmos DB for MongoDB vCore (June 2023)



- **Dados de origem:** é aqui que os dados existem. Pode ser um arquivo/pasta em seu computador, um arquivo no armazenamento em nuvem, um ativo de dados do Azure Machine Learning, um repositório Git ou um banco de dados SQL.
- **Partes de dados:** os dados na sua fonte precisam ser convertidos em texto sem formatação. Por exemplo, documentos do Word ou PDFs precisam ser abertos e convertidos em texto. Em seguida, o texto é agrupado em partes menores.
- **Converter o texto em vetores:** são as inserções. Os vetores são representações numéricas de conceitos convertidos em sequências numéricas, o que torna mais fácil para os computadores entenderem as relações entre esses conceitos.
- **Links entre dados de origem e inserções:** essas informações são armazenadas como metadados nas partes criadas, que são usadas para ajudar os LLMs a gerar citações ao gerar respostas.

Conteúdo relacionado

- [Engenharia de prompts](#)

Noções básicas sobre a chamada de função do OpenAI

Artigo • 27/05/2024

A chamada de função é um recurso de modelo do OpenAI que permite descrever as funções e seus argumentos nos prompts usando o JSON. Em vez de invocar a função em si, o modelo retorna uma saída JSON que descreve quais funções devem ser chamadas e os argumentos a serem usados.

A chamada de função simplifica como você conecta ferramentas externas ao seu modelo de IA. Primeiro, especifique as funções de cada ferramenta para o modelo. Em seguida, o modelo decide quais funções devem ser chamadas, com base na pergunta de prompt. O modelo usa os resultados da chamada de função para criar uma resposta mais precisa e consistente.

Os possíveis casos de uso para chamadas de função incluem:

- Responder perguntas chamando APIs externas, por exemplo, enviando emails ou obtendo a previsão do tempo.
- Responder perguntas com informações de um armazenamento de dados interno, por exemplo, agregando dados de vendas para responder: "Quais são meus produtos mais vendidos?".
- Criar dados estruturados a partir de informações de texto, por exemplo, criando um objeto de informações do usuário com detalhes do histórico de chat.

Chamar funções com o OpenAI

As etapas gerais para chamar funções com um modelo do OpenAI são:

1. Envie a pergunta do usuário como uma solicitação com funções definidas nos [parâmetros tools](#).
2. O modelo decide quais funções, se houver, devem ser chamadas. A saída contém um objeto JSON que lista as chamadas de função e seus argumentos.

⚠ Observação

O modelo pode alucinar argumentos adicionais.

3. Analise a saída e chame as funções solicitadas com seus argumentos especificados.

4. Envie outra solicitação com os resultados da função incluídos como uma nova mensagem.
5. O modelo responde com mais solicitações de chamada de função ou uma resposta à pergunta do usuário.
 - Continue invocando as chamadas de função solicitadas até que o modelo responda com uma resposta.

Você pode forçar o modelo a solicitar uma função específica definindo o parâmetro `tool_choice` como o nome da função. Você também pode forçar o modelo a responder com uma mensagem para o usuário definindo o parâmetro `tool_choice` como `"none"`.

Chamar funções paralelamente

Alguns modelos são compatíveis com a chamada de função paralela, o que permite que o modelo solicite várias chamadas de função em uma saída. Os resultados de cada chamada de função são incluídos juntos em uma resposta ao modelo. A chamada de função paralela reduz o número de solicitações de API e o tempo necessários para gerar uma resposta. Cada resultado da função é incluído como uma nova mensagem na conversa com um `tool_call_id` correspondendo ao `id` da solicitação de chamada de função.

Modelos com suporte

Nem todos os modelos do OpenAI são treinados para dar suporte à chamada de função. Para obter uma lista de modelos compatíveis com a chamada de função ou a chamada de função paralela, confira [OpenAI – Modelos compatíveis](#).

Chamada de função com o SDK do Kernel Semântico

O [SDK do Kernel Semântico](#) dá suporte à descrição de quais funções estão disponíveis para sua IA usando o decorador `KernelFunction`.

O Kernel cria o parâmetro `tools` de uma solicitação com base nos decoradores, orquestra as chamadas de função solicitadas ao seu código e retorna os resultados de volta ao modelo.

Contagens de tokens

As descrições de função são incluídas na mensagem do sistema da solicitação para um modelo. Essas descrições de função são contabilizadas no [limite de token](#) do modelo e estão [incluídas no custo da solicitação](#).

Se a solicitação exceder o limite de token do modelo, tente as seguintes modificações:

- Reduza o número de funções.
- Reduza as descrições de função e o argumento no JSON.

Conteúdo relacionado

- [Noções básicas de tokens](#)
- [Criação de funções nativas para a IA chamar](#)
- [Engenharia de prompts](#)

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).

.NET

Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 [Abrir um problema de documentação](#)

 [Fornecer comentários sobre o produto](#)

Autenticar e autorizar o Serviço de Aplicativo no OpenAI do Azure usando o Microsoft Entra e o SDK do Kernel Semântico

Artigo • 27/05/2024

Este artigo demonstra como usar as [identidades gerenciadas pelo Microsoft Entra](#) para autenticar e autorizar um aplicativo do Serviço de Aplicativo para um recurso do OpenAI do Azure.

Este artigo também demonstra como usar o [SDK do Kernel Semântico](#) para implementar facilmente a autenticação do Microsoft Entra no seu aplicativo .NET.

Usando uma identidade gerenciada do Microsoft Entra, seu aplicativo do Serviço de Aplicativo pode acessar facilmente os recursos protegidos do OpenAI do Azure sem a necessidade de provisionar ou girar segredos manualmente.

Pré-requisitos

- Uma conta do Azure com uma assinatura ativa. [Crie uma conta gratuitamente](#).
- [SDK .NET](#)
- [Microsoft.SemanticKernel Pacote NuGet](#)
- [Pacote NuGet Azure.Identity](#)
- [Criar e implantar um recurso do Serviço OpenAI do Azure](#)
- [Criar e implantar um aplicativo .NET no Serviço de Aplicativo](#)

Adicionar uma identidade gerenciada ao Serviço de Aplicativo

Seu aplicativo pode receber dois tipos de identidades:

- Uma **identidade atribuída pelo sistema** é vinculada ao seu aplicativo e é excluída se o seu aplicativo for excluído. Um aplicativo pode ter apenas uma identidade atribuída pelo sistema.
- Uma **identidade atribuída pelo usuário** é um recurso independente do Azure que pode ser atribuído ao seu aplicativo. Um aplicativo pode ter várias identidades atribuídas pelo usuário.

Adicionar uma identidade atribuída pelo sistema

1. Navegue até a página do aplicativo no [portal do Azure](#) e role para baixo até o grupo **Configurações**.
2. Selecionar **Identidade**.
3. Na guia **Atribuído pelo Sistema**, alterne *Status* para **Ativado** e selecione **Salvar**.

Adicionar uma identidade atribuída pelo usuário

Para adicionar uma identidade atribuída pelo usuário ao seu aplicativo, crie a identidade e adicione seu identificador de recurso à configuração do aplicativo.

1. Crie um recurso de identidade gerenciada atribuída pelo usuário seguindo [estas instruções](#).
2. No painel de navegação esquerdo da página do aplicativo, role para baixo até o grupo **Configurações**.
3. Selecionar **Identidade**.
4. Selecione **Usuário atribuído > Adicionar**.
5. Localize a identidade que você criou anteriormente, selecione-a e clique em **Adicionar**.

 **Importante**

Depois de selecionar **Adicionar**, o aplicativo será reiniciado.

Adicionar uma função de usuário do OpenAI do Azure à sua identidade gerenciada

1. No [portal do Azure](#), navegue até o escopo ao qual deseja conceder acesso ao **OpenAI do Azure**. O escopo pode ser um **Grupo de gerenciamento**, **Assinatura**, **Grupo de recursos** ou um recurso específico do **OpenAI do Azure**.
2. No painel de navegação à esquerda, selecione **Controle de acesso (IAM)**.
3. Selecione **Adicionar** e **Adicionar atribuição de função**.
4. Na guia **Função**, selecione a função **Usuário do OpenAI dos Serviços Cognitivos**.
5. Na guia **Membros**, selecione a identidade gerenciada.
6. Na guia **Examinar + atribuir**, selecione **Examinar + atribuir** para atribuir a função.

Implementar a autenticação baseada em token usando o SDK do Kernel Semântico

1. Initialize um objeto `DefaultAzureCredential` para suportar a identidade gerenciada do aplicativo:

```
C#  
  
// Initialize a DefaultAzureCredential.  
// This credential type will try several authentication flows in order  
// until one is available.  
// Will pickup Visual Studio or Azure CLI credentials in local  
// environments.  
// Will pickup managed identity credentials in production deployments.  
TokenCredential credentials = new DefaultAzureCredential(  
    new DefaultAzureCredentialOptions  
    {  
        // If using a user-assigned identity specify either:  
        // ManagedIdentityClientId or ManagedIdentityResourceId.  
        // e.g.: ManagedIdentityClientId = "myIdentityClientId".  
    }  
);
```

2. Crie um objeto `Kernel` que inclua o Serviço de Conclusão de Chat do OpenAI do Azure e use as credenciais criadas anteriormente:

```
C#  
  
// Retrieve the endpoint and deployment obtained from the Azure OpenAI  
// deployment.  
// Must use the deployment name not the underlying model name.  
IConfigurationRoot config = new  
ConfigurationBuilder().AddUserSecrets<Program>().Build();  
string endpoint = config["AZURE_OPENAI_ENDPOINT"]!;  
string deployment = config["AZURE_OPENAI_GPT_NAME"]!;  
  
// Build a Kernel that includes the Azure OpenAI Chat Completion  
// Service.  
// Include the previously created token credential.  
Kernel kernel = Kernel  
    .CreateBuilder()  
    .AddAzureOpenAIChatCompletion(deployment, endpoint, credentials)  
    .Build();
```

3. Use o objeto `Kernel` para invocar a conclusão do prompt por meio do OpenAI do Azure:

```
C#
```

```
// Use the Kernel to invoke prompt completion through Azure OpenAI.  
// The Kernel response will be null if the model can't be reached.  
string? result = await kernel.InvokePromptAsync<string>("Please list  
three Azure services");  
Console.WriteLine($"Output: {result}");  
  
// Continue sending and receiving messages between the user and AI.  
// ...
```

Conteúdo relacionado

- Autenticar e autorizar o Serviço de Aplicativo em um banco de dados vetorial
- Como usar identidades gerenciadas para o Serviço de Aplicativo e o Azure Functions
- Controle de acesso baseado em função para o Serviço OpenAI do Azure

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).

.NET

Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 [Abrir um problema de documentação](#)

 [Fornecer comentários sobre o produto](#)

Autenticar e autorizar o Serviço de Aplicativo em um banco de dados vetorial

Artigo • 26/05/2024

Este artigo demonstra como gerenciar a conexão entre o aplicativo .NET do Serviço de Aplicativo e uma [solução de banco de dados vetorial](#). Ele aborda o uso de identidades gerenciadas do Microsoft Entra para serviços com suporte e o armazenamento seguro de cadeias de conexão para outras pessoas.

Ao adicionar um banco de dados vetorial ao seu aplicativo, você pode habilitar [memórias semânticas](#) para sua IA. O [SDK de Kernel Semântico](#) para .NET permite que você implemente facilmente o armazenamento de memória e o recall usando sua solução de banco de dados vetorial preferencial.

Pré-requisitos

- Uma conta do Azure com uma assinatura ativa. [Crie uma conta gratuitamente](#).
- [SDK .NET](#)
- [Pacote NuGet Microsoft.SemanticKernel](#)
- [Pacote NuGet Microsoft.SemanticKernel.Plugins.Memory](#)
- [Criar e implantar um aplicativo .NET no Serviço de Aplicativo](#)
- [Criar e implantar uma solução de banco de dados vetorial](#)

Usar identidade gerenciada do Microsoft Entra para autenticação

Se um serviço de banco de dados vetorial der suporte à autenticação do Microsoft Entra, você poderá usar uma identidade gerenciada com o Serviço de Aplicativo para acessar com segurança seu banco de dados vetorial sem precisar provisionar ou girar manualmente os segredos. Para obter uma lista dos serviços do Azure que dão suporte à autenticação do Microsoft Entra, consulte [Serviços do Azure que dão suporte à autenticação do Microsoft Entra](#).

Adicionar uma identidade gerenciada ao Serviço de Aplicativo

Seu aplicativo pode receber dois tipos de identidades:

- Uma **identidade atribuída pelo sistema** é vinculada ao seu aplicativo e é excluída se o seu aplicativo for excluído. Um aplicativo pode ter apenas uma identidade atribuída pelo sistema.
- Uma **identidade atribuída pelo usuário** é um recurso independente do Azure que pode ser atribuído ao seu aplicativo. Um aplicativo pode ter várias identidades atribuídas pelo usuário.

Adicionar uma identidade atribuída pelo sistema

1. Navegue até a página do aplicativo no [portal do Azure](#) e role para baixo até o grupo **Configurações**.
2. Selecionar **Identidade**.
3. Na guia **Atribuído pelo Sistema**, alterne *Status* para **Ativado** e selecione **Salvar**.

Adicionar uma identidade atribuída pelo usuário

Para adicionar uma identidade atribuída pelo usuário ao seu aplicativo, crie a identidade e adicione seu identificador de recurso à configuração do aplicativo.

1. Crie um recurso de identidade gerenciada atribuída pelo usuário seguindo [estas instruções](#).
2. No painel de navegação esquerdo da página do aplicativo, role para baixo até o grupo **Configurações**.
3. Selecionar **Identidade**.
4. Selecione **Usuário atribuído > Adicionar**.
5. Localize a identidade que você criou anteriormente, selecione-a e clique em **Adicionar**.

Importante

Depois de selecionar **Adicionar**, o aplicativo será reiniciado.

Adicionar uma função do Azure à sua identidade gerenciada

1. No [portal do Azure](#), navegue até o escopo ao qual você deseja conceder acesso ao banco de dados vetorial. O escopo pode ser um **Grupo de gerenciamento**, **Assinatura**, **Grupo de recursos** ou um recurso específico do Azure.
2. No painel de navegação à esquerda, selecione **Controle de acesso (IAM)**.
3. Selecione **Adicionar e Adicionar atribuição de função**.
4. Na guia **Função**, selecione a função apropriada que concede acesso de leitura ao banco de dados vetorial.
5. Na guia **Membros**, selecione a identidade gerenciada.
6. Na guia **Examinar + atribuir**, selecione **Examinar + atribuir** para atribuir a função.

Implementar a autenticação baseada em token com o banco de dados vetorial

Os exemplos de código a seguir exigem estas bibliotecas adicionais:

- [Pacote NuGet Azure.Identity](#)
- [Pacote NuGet Microsoft.SemanticKernel.Connectors.AzureAISeach](#)

1. Inicialize um objeto `DefaultAzureCredential` para obter a identidade gerenciada do aplicativo:

```
C#  
  
// Initialize a DefaultAzureCredential.  
// This credential type will try several authentication flows in order  
// until one is available.  
// Will pickup Visual Studio or Azure CLI credentials in local  
// environments.  
// Will pickup managed identity credentials in production deployments.  
TokenCredential credentials = new DefaultAzureCredential(  
    new DefaultAzureCredentialOptions  
    {  
        // If using a user-assigned identity specify either:  
        // ManagedIdentityClientId or ManagedIdentityResourceId.  
        // e.g.: ManagedIdentityClientId = "myIdentityClientId".  
    }  
);
```

2. Inicialize um objeto `IMemoryStore` para o banco de dados vetorial e, em seguida, use-o para criar um `ISemanticTextMemory`:

```
C#  
  
// Retrieve the endpoint obtained from the Azure AI Search deployment.  
// Retrieve the endpoint and deployments obtained from the Azure OpenAI  
// deployment.
```

```

// Must use the deployment name not the underlying model name.
IConfigurationRoot config = new
ConfigurationBuilder().AddUserSecrets<Program>().Build();
string searchEndpoint = config["AZURE_AISEARCH_ENDPOINT"]!;
string openAiEndpoint = config["AZURE_OPENAI_ENDPOINT"]!;
string embeddingModel = config["AZURE_OPENAI_EMBEDDING_NAME"]!;

// The Semantic Kernel SDK provides a connector extension for Azure AI
Search.
// Initialize an AzureAISeachMemoryStore using your managed identity
credentials.
IMemoryStore memoryStore = new AzureAISeachMemoryStore(searchEndpoint,
credentials);

// Build a SemanticMemoryStore with Azure AI Search as the store.
// Must also include a text embedding generation service.
ISemanticTextMemory memory = new MemoryBuilder()
    .WithAzureOpenAITextEmbeddingGeneration(embeddingModel,
openAiEndpoint, credentials)
    .WithMemoryStore(memoryStore)
    .Build();

```

3. Crie um objeto `Kernel` e depois importe o objeto `ISemanticTextMemory` usando o `TextMemoryPlugin`:

C#

```

// Build a Kernel, include a chat completion service.
string chatModel = config["AZURE_OPENAI_GPT_NAME"]!;
Kernel kernel = Kernel
    .CreateBuilder()
    .AddAzureOpenAIChatCompletion(chatModel, openAiEndpoint,
credentials)
    .Build();

// Import the semantic memory store as a TextMemoryPlugin.
// The TextMemoryPlugin enable recall via prompt expressions.
kernel.ImportPluginFromObject(new TextMemoryPlugin(memory));

```

4. Use o objeto `Kernel` para invocar um prompt que inclua o recall de memória:

C#

```

// Must configure the memory collection, number of memories to recall,
and relevance score.
// The {{...}} syntax represents an expression to Semantic Kernel.
// For more information on this syntax see:
// https://learn.microsoft.com/semantic-kernel/prompts/prompt-template-
syntax
string memoryCollection = config["AZURE_OPENAI_MEMORY_NAME"]!;
string? result = await kernel.InvokePromptAsync<string>(

```

```
"{{recall 'where did I grow up?'}}",  
new()  
{  
    [TextMemoryPlugin.CollectionParam] = memoryCollection,  
    [TextMemoryPlugin.LimitParam] = "2",  
    [TextMemoryPlugin.RelevanceParam] = "0.79",  
}  
};  
Console.WriteLine($"Output: {result}");
```

Usar o Key Vault para armazenar segredos de conexão

Se um banco de dados vetorial não der suporte à autenticação do Microsoft Entra, você poderá usar um Key Vault para armazenar seus segredos de conexão e recuperá-los com seu aplicativo do Serviço de Aplicativo. Ao usar um Key Vault para armazenar seus segredos de conexão, você pode compartilhá-los com vários aplicativos e controlar o acesso a segredos individuais por aplicativo.

Antes de seguir estas etapas, recupere uma cadeia de conexão para o banco de dados vetorial. Por exemplo, consulte [Usar o Cache do Azure para Redis com um aplicativo web ASP.NET Core](#).

Adicionar uma cadeia de conexão ao Key Vault

ⓘ Importante

Antes de seguir estas etapas, verifique se você [criou um Key Vault usando o portal do Azure](#).

1. Navegue até seu cofre de chaves no [portal do Azure](#).
2. Na navegação à esquerda do Key Vault, selecione **Objetos** e selecione **Segredos**.
3. Selecione + Gerar/importar.
4. Na tela **Criar um segredo**, escolha os seguintes valores:
 - **Opções de upload:** Manual.
 - **Nome:** Digite um nome para o segredo. O nome do segredo precisa ser exclusivo dentro de um Key Vault.
 - **Valor:** a cadeia de conexão do banco de dados vetorial.
 - Deixe os outros valores com seus padrões. Selecione **Criar**.

5. Quando você recebe a mensagem de que o segredo foi criado com sucesso, ele está pronto para ser usado em seu aplicativo.

Conceder ao Serviço de Aplicativo acesso ao Key Vault

1. Atribuir uma identidade gerenciada ao seu Serviço de Aplicativo.
2. Adicione as funções Key Vault Secrets User e Key Vault Reader à sua identidade gerenciada.

Implementar a recuperação de cadeia de conexão do Key Vault

Para usar os exemplos de código a seguir, você precisa destas bibliotecas adicionais:

- Pacote NuGet Azure.Identity ↗
- Pacote NuGet Azure.Extensions.AspNetCore.Configuration.Secrets ↗
- Pacote NuGet Microsoft.Extensions.Configuration ↗

Esses exemplos de código usam um banco de dados Redis, mas você pode aplicá-los a qualquer banco de dados vetorial que dê suporte a cadeias de conexão.

1. Inicialize um objeto `DefaultAzureCredential` para obter a identidade gerenciada do aplicativo:

```
C#  
  
// Initialize a DefaultAzureCredential.  
// This credential type will try several authentication flows in order  
// until one is available.  
// Will pickup Visual Studio or Azure CLI credentials in local  
// environments.  
// Will pickup managed identity credentials in production deployments.  
TokenCredential credentials = new DefaultAzureCredential(  
    new DefaultAzureCredentialOptions  
    {  
        // If using a user-assigned identity specify either:  
        // ManagedIdentityClientId or ManagedIdentityResourceId.  
        // e.g.: ManagedIdentityClientId = "myIdentityClientId".  
    }  
);
```

2. Adicione o Key Vault ao compilar sua configuração, isso mapeará seus segredos do Key Vault para o objeto `IConfigurationRoot`:

```
C#
```

```

// User secrets let you provide connection strings when testing locally
// For more info see: https://learn.microsoft.com/en-
us/aspnet/core/security/app-secrets
IConfigurationRoot config = new ConfigurationBuilder()
    .AddUserSecrets<Program>()
    .AddAzureKeyVault(new Uri("{vaultURI}"), credentials)
    .Build();

// Retrieve the Redis connection string obtained from the Key Vault.
string redisConnectionString = config["AZURE_REDIS_CONNECT_STRING"]!;

```

3. Use a cadeia de conexão de banco de dados vetorial do Key Vault para inicializar um objeto `IMemoryStore` e, em seguida, use-a para criar um `ISemanticTextMemory`:

C#

```

// Use the connection string to connect to the database
IDatabase database = (
    await ConnectionMultiplexer.ConnectAsync(redisConnectionString)
).GetDatabase();

// The Semantic Kernel SDK provides a connector extension for Redis.
// Initialize an RedisMemoryStore using your managed identity
// credentials.
IMemoryStore memoryStore = new RedisMemoryStore(database);

// Retrieve the endpoint and deployments obtained from the Azure OpenAI
// deployment.
// Must use the deployment name not the underlying model name.
string openAiEndpoint = config["AZURE_OPENAI_ENDPOINT"]!;
string embeddingModel = config["AZURE_OPENAI_EMBEDDING_NAME"]!;

// Build a SemanticMemoryStore with Azure AI Search as the store.
// Must also include a text embedding generation service.
ISemanticTextMemory memory = new MemoryBuilder()
    .WithAzureOpenAITextEmbeddingGeneration(embeddingModel,
openAiEndpoint, credentials)
    .WithMemoryStore(memoryStore)
    .Build();

```

4. Crie um objeto `Kernel` e depois importe o objeto `ISemanticTextMemory` usando o `TextMemoryPlugin`:

C#

```

// Build a Kernel, include a chat completion service.
string chatModel = config["AZURE_OPENAI_GPT_NAME"]!;
Kernel kernel = Kernel
    .CreateBuilder()
    .AddAzureOpenAIChatCompletion(chatModel, openAiEndpoint,

```

```
credentials)
    .Build();

// Import the semantic memory store as a TextMemoryPlugin.
// The TextMemoryPlugin enable recall via prompt expressions.
kernel.ImportPluginFromObject(new TextMemoryPlugin(memory));
```

5. Use o objeto `Kernel` para invocar um prompt que inclua o recall de memória:

C#

```
// Must configure the memory collection, number of memories to recall,
// and relevance score.
// The {{...}} syntax represents an expression to Semantic Kernel.
// For more information on this syntax see:
// https://learn.microsoft.com/semantic-kernel/prompts/prompt-template-
// syntax
string memoryCollection = config["AZURE_OPENAI_MEMORY_NAME"]!;
string? result = await kernel.InvokePromptAsync<string>(
    "{{recall 'where did I grow up?'}}",
    new()
{
    [TextMemoryPlugin.CollectionParam] = memoryCollection,
    [TextMemoryPlugin.LimitParam] = "2",
    [TextMemoryPlugin.RelevanceParam] = "0.79",
}
);
Console.WriteLine($"Output: {result}");
```

Usar configurações de aplicativo para armazenar segredos de conexão

Se um banco de dados vetorial não der suporte à autenticação do Microsoft Entra, você pode usar as configurações do aplicativo do Serviço de Aplicativo para armazenar seus segredos de conexão. Ao usar configurações de aplicativo, você pode armazenar seus segredos de conexão sem provisionar recursos adicionais do Azure.

Antes de seguir estas etapas, recupere uma cadeia de conexão para o banco de dados vetorial. Por exemplo, consulte [Usar o Cache do Azure para Redis no .NET Framework](#).

Adicionar uma cadeia de conexão às configurações do aplicativo

1. Navegue até a página do seu aplicativo no [portal do Azure](#).

2. No menu à esquerda do aplicativo, selecione **Configurações**>**Configurações do aplicativo**.

- Por padrão, os valores das configurações do aplicativo ficam ocultos no portal para segurança.
- Para ver um valor oculto de uma configuração de aplicativo, selecione o campo **Valor**.

3. Selecione **Nova configuração de conexão**.

4. Na tela **Adicionar/Editar cadeia de conexão**, escolha os seguintes valores:

- **Nome**: digite um nome para a configuração. O nome da configuração precisa ser exclusivo.
- **Valor**: a cadeia de conexão do banco de dados vetorial.
- **Tipo**: o tipo de conexão, **Custom** se nenhum outro se aplicar.
- Deixe os outros valores com seus padrões. Selecione **OK**.

5. Selecione **Salvar** de volta na página de Configuração.

Implementar a recuperação de cadeia de conexão das configurações do aplicativo

Para usar os exemplos de código a seguir, você precisa destas bibliotecas adicionais:

- Pacote NuGet [Microsoft.Extensions.Configuration](#)
- Pacote NuGet [Microsoft.Extensions.Configuration.EnvironmentVariables](#)

Esses exemplos de código usam um banco de dados Redis, mas você pode aplicá-los a qualquer banco de dados vetorial que dê suporte a cadeias de conexão.

1. Adicione variáveis de ambiente ao criar sua configuração, isso mapeará suas cadeias de conexão para o objeto **IConfigurationRoot**:

```
C#  
  
// User secrets let you provide connection strings when testing locally  
// For more info see: https://learn.microsoft.com/en-us/aspnet/core/security/app-secrets  
IConfigurationRoot config = new ConfigurationBuilder()  
    .AddUserSecrets<Program>()  
    .AddEnvironmentVariables()  
    .Build();  
  
// Retrieve the Redis connection string obtained from the app settings.  
// The connection string name should match the entry in application  
settings
```

```
string redisConnectionString =
config.GetConnectionString("AZURE_REDIS")!;
```

2. Use a cadeia de conexão de banco de dados vetorial da configurações do aplicativo para inicializar um objeto `IMemoryStore` e, em seguida, use-a para criar um `ISemanticTextMemory`:

C#

```
// Use the connection string to connect to the database
IDatabase database = (
    await ConnectionMultiplexer.ConnectAsync(redisConnectionString)
).GetDatabase();

// The Semantic Kernel SDK provides a connector extension for Redis.
// Initialize an RedisMemoryStore using your managed identity
// credentials.
IMemoryStore memoryStore = new RedisMemoryStore(database);

// Retrieve the endpoint and deployments obtained from the Azure OpenAI
// deployment.
// Must use the deployment name not the underlying model name.
string openAiEndpoint = config["AZURE_OPENAI_ENDPOINT"]!;
string embeddingModel = config["AZURE_OPENAI_EMBEDDING_NAME"]!

// Build a SemanticMemoryStore with Azure AI Search as the store.
// Must also include a text embedding generation service.
ISemanticTextMemory memory = new MemoryBuilder()
    .WithAzureOpenAITextEmbeddingGeneration(embeddingModel,
openAiEndpoint, credentials)
    .WithMemoryStore(memoryStore)
    .Build();
```

3. Crie um objeto `Kernel` e depois importe o objeto `ISemanticTextMemory` usando o `TextMemoryPlugin`:

C#

```
// Build a Kernel, include a chat completion service.
string chatModel = config["AZURE_OPENAI_GPT_NAME"]!;
Kernel kernel = Kernel
    .CreateBuilder()
    .AddAzureOpenAIChatCompletion(chatModel, openAiEndpoint,
credentials)
    .Build();

// Import the semantic memory store as a TextMemoryPlugin.
// The TextMemoryPlugin enable recall via prompt expressions.
kernel.ImportPluginFromObject(new TextMemoryPlugin(memory));
```

4. Use o objeto `Kernel` para invocar um prompt que inclua o recall de memória:

```
C#  
  
// Must configure the memory collection, number of memories to recall,  
and relevance score.  
// The {{...}} syntax represents an expression to Semantic Kernel.  
// For more information on this syntax see:  
// https://learn.microsoft.com/semantic-kernel/prompts/prompt-template-  
syntax  
string memoryCollection = config["AZURE_OPENAI_MEMORY_NAME"]!;  
string? result = await kernel.InvokePromptAsync<string>(  
    "{{recall 'where did I grow up?'}}",  
    new()  
{  
    [TextMemoryPlugin.CollectionParam] = memoryCollection,  
    [TextMemoryPlugin.LimitParam] = "2",  
    [TextMemoryPlugin.RelevanceParam] = "0.79",  
}  
);  
Console.WriteLine($"Output: {result}");
```

Conteúdo relacionado

- [Use o Redis para armazenamento de memória com o SDK de Kernel Semântico]
- [Como usar identidades gerenciadas para o Serviço de Aplicativo e o Azure Functions](#)
- [Etapas para atribuir funções no Azure](#)

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).

.NET

Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 [Abrir um problema de documentação](#)

 [Fornecer comentários sobre o produto](#)

Use o Redis para armazenamento de memória com o SDK do Semantic Kernel

Artigo • 04/10/2024

Este artigo demonstra como integrar um banco de dados Redis com o módulo RediSearch no [SDK do Semantic Kernel](#) e usá-lo para armazenamento e recuperação de memória.

Os [repositórios de vetores](#) representam informações de texto que foram armazenadas junto com um vetor de incorporação pré-computado para o texto inteiro. Quando um LLM é solicitado a se lembrar de uma memória, ele usa essas inserções pré-computadas para avaliar com eficiência se uma memória é relevante para o prompt. Depois que o LLM encontra uma memória correspondente, ele usa as informações de texto da memória como contexto para as próximas etapas da conclusão do prompt.

O armazenamento de memória adicionado ao SDK do Semantic Kernel fornece um contexto mais amplo para suas solicitações. Ele também permite que você armazene dados da mesma maneira que armazena um banco de dados tradicional, mas consulte-os utilizando linguagem natural.

Pré-requisitos

- Uma conta do Azure com uma assinatura ativa. [Crie uma conta gratuitamente](#).
- [SDK .NET](#)
- [Microsoft.SemanticKernel Pacote NuGet](#)
- [Microsoft.SemanticKernel.Connectors.Redis Pacote NuGet](#)
- [Microsoft.SemanticKernel.Plugins.Memory Pacote NuGet](#)
- [StackExchange.Redis Pacote NuGet](#)
- Um banco de dados Redis que tem o módulo RediSearch, [implantado e acessível ao seu aplicativo .NET](#)

Implementar o armazenamento de memória usando um banco de dados Redis

Antes de integrar seu banco de dados Redis ao SDK do Semantic Kernel, certifique-se de que o módulo RediSearch esteja habilitado. Para obter informações sobre o módulo *Cache do Azure para Redis*, consulte [Usar módulos Redis com o Cache do Azure para Redis](#).

1. Inicializar uma conexão com seu banco de dados Redis. Por exemplo:

```
C#  
  
// Retrieve the Redis connection config.  
IConfigurationRoot config = new  
ConfigurationBuilder().AddUserSecrets<Program>().Build();  
string redisConfig = config["REDIS_CONFIG"]!;  
  
// Initialize a connection to the Redis database.  
ConnectionMultiplexer connectionMultiplexer = await  
ConnectionMultiplexer.ConnectAsync(  
    redisConfig  
);  
IDatabase database = connectionMultiplexer.GetDatabase();
```

2. Crie o `Kernel` incluindo o `ITextEmbeddingGenerationService`. Por exemplo:

```
C#  
  
// Retrieve the Azure OpenAI config and secrets saved during  
deployment.  
string endpoint = config["AZURE_OPENAI_ENDPOINT"]!;  
string embeddingModel = config["AZURE_OPENAI_EMBEDDING_NAME"]!;  
string completionModel = config["AZURE_OPENAI_GPT_NAME"]!;  
string key = config["AZURE_OPENAI_KEY"]!;  
  
// Build the Kernel; must add an embedding generation service.  
Kernel kernel = Kernel  
    .CreateBuilder()  
    .AddAzureOpenAITextEmbeddingGeneration(embeddingModel, endpoint,  
key)  
    .AddAzureOpenAIChatCompletion(completionModel, endpoint, key)  
    .Build();
```

3. Envolva o banco de dados Redis em uma instância `RedisMemoryStore` e, em seguida, inicialize um objeto `SemanticTextMemory` usando o armazenamento de memória e o serviço de geração de inserção. Por exemplo:

```
C#  
  
// Retrieve the desired vector size for the memory store.  
// If unspecified, the default vector size is 1536.  
int vectorSize = int.Parse(config["REDIS_MEMORY_VECTOR_SIZE"]!);  
  
// Initialize a memory store using the redis database  
IMemoryStore memoryStore = new RedisMemoryStore(database, vectorSize);  
  
// Retrieve the embedding service from the Kernel.  
ITextEmbeddingGenerationService embeddingService =
```

```

        kernel.Services.GetRequiredService<ITextEmbeddingGenerationService>
    ());

    // Initialize a SemanticTextMemory using the memory store and embedding
    // generation service.
    SemanticTextMemory textMemory = new(memoryStore, embeddingService);

```

4. Adicione a memória de texto semântico ao `Kernel` usando a classe `TextMemoryPlugin`. Por exemplo:

C#

```

// Initialize a TextMemoryPlugin using the text memory.
TextMemoryPlugin memoryPlugin = new(textMemory);

// Import the text memory plugin into the Kernel.
KernelPlugin memory = kernel.ImportPluginFromObject(memoryPlugin);

```

5. Use o `Kernel` e o plug-in para salvar, recuperar e fazer recall de memórias. Por exemplo:

C#

```

// Retrieve the desired memory collection name.
string memoryCollectionName = config["REDIS_MEMORY_COLLECTION_NAME"]!;

// Save a memory with the Kernel.
await kernel.InvokeAsync(
    memory["Save"],
    new()
    {
        [TextMemoryPlugin.InputParam] = "My family is from New York",
        [TextMemoryPlugin.CollectionParam] = memoryCollectionName,
        [TextMemoryPlugin.KeyParam] = "info1",
    }
);

// Retrieve a memory with the Kernel.
FunctionResult result = await kernel.InvokeAsync(
    memory["Retrieve"],
    new()
    {
        [TextMemoryPlugin.CollectionParam] = memoryCollectionName,
        [TextMemoryPlugin.KeyParam] = "info1",
    }
);

// Get the memory string from the function result; returns a null value
// if no memory is found.
Console.WriteLine(
    $"Retrieved memory: {result.GetValue<string>()} ?? "ERROR: memory

```

```

    not found}"}

);

// Alternatively, recall similar memories with the Kernel.
// Can configure the memory collection, number of memories to recall,
// and relevance score.

result = await kernel.InvokeAsync(
    memory["Recall"],
    new()
    {
        [TextMemoryPlugin.InputParam] = "Ask: where do I live?",
        [TextMemoryPlugin.CollectionParam] = memoryCollectionName,
        [TextMemoryPlugin.LimitParam] = "2",
        [TextMemoryPlugin.RelevanceParam] = "0.79",
    }
);

// If memories are recalled, the function result can be deserialized as
// a string[].

string? resultStr = result.GetValue<string>();
string[]? parsedResult = string.IsNullOrEmpty(resultStr)
    ? null
    : JsonSerializer.Deserialize<string[]>(resultStr);
Console.WriteLine(
    $"Recalled memories: {(parsedResult?.Length > 0 ? resultStr :
    "ERROR: memory not found")}"
);

```

6. Use o recall de memória como parte de um prompt usando a sintaxe do modelo de prompt `{...}`. Por exemplo:

```

C#


// Create a prompt that includes memory recall.
// The {{...}} syntax represents an expression to Semantic Kernel.
// For more information on this syntax see:
// https://learn.microsoft.com/semantic-kernel/prompts/prompt-template-syntax
string memoryRecallPrompt = """
    Consider only the facts below when answering questions:

    BEGIN FACTS
    About me: {{recall 'where did I grow up?'}}
    END FACTS

    Question: What are some fun facts about my home state?
    """;

// Invoke the prompt with the Kernel.
// Must configure the memory collection, number of memories to recall,
// and relevance score.
resultStr = await kernel.InvokePromptAsync<string>(
    memoryRecallPrompt,

```

```
new()
{
    [TextMemoryPlugin.CollectionParam] = memoryCollectionName,
    [TextMemoryPlugin.LimitParam] = "2",
    [TextMemoryPlugin.RelevanceParam] = "0.79",
}
);

// If the memory recall fails, the model will indicate it has missing
information in its output.
// Otherwise the output will incorporate your memory as context.
Console.WriteLine($"Output: {resultStr}");
```

Conteúdo relacionado

- [Usar RAG com SQL]
- [Ingestão de dados do SharePoint]
- [Trabalhando com bancos de dados vetoriais]

Usar modelos de IA personalizados e locais com o SDK do Kernel Semântico

Artigo • 27/05/2024

Este artigo demonstra como integrar modelos personalizados e locais ao [SDK do Kernel Semântico](#) e usá-los para geração de texto e conclusões de chat.

Você pode adaptar as etapas para usá-las com qualquer modelo que possa acessar, independentemente do local ou modo de acesso. Por exemplo, você pode integrar o modelo de [codellama](#) ao SDK do Kernel Semântico para habilitar a geração e a discussão de código.

Geralmente, os modelos personalizados e locais fornecem acesso por meio de APIs REST, por exemplo, confira [Compatibilidade do Ollama OpenAI](#). Antes de integrar o modelo, ele precisará ser hospedado e estar acessível ao seu aplicativo do .NET por meio de HTTPS.

Pré-requisitos

- Uma conta do Azure com uma assinatura ativa. [Crie uma conta gratuitamente](#).
- [SDK .NET](#).
- [Microsoft.SemanticKernel Pacote NuGet](#)
- Um modelo personalizado ou local, implantado e acessível ao aplicativo do .NET

Implementar a geração de texto usando um modelo local

A seção a seguir mostra como você pode integrar seu modelo ao SDK do Kernel Semântico e usá-lo para gerar conclusões de texto.

1. Crie uma classe de serviço que implemente a interface `ITextGenerationService`.

Por exemplo:

```
C#  
  
class MyTextGenerationService : ITextGenerationService  
{  
    private IReadOnlyDictionary<string, object?>? _attributes;  
    public IReadOnlyDictionary<string, object?> Attributes =>  
        _attributes ??= new Dictionary<string, object?>();
```

```
    public string ModelUrl { get; init; } = "<default url to your
model's Chat API>";
    public required string ModelApiKey { get; init; }

    public async IAsyncEnumerable<StreamingTextContent>
GetStreamingTextContentsAsync(
        string prompt,
        PromptExecutionSettings? executionSettings = null,
        Kernel? kernel = null,
        [EnumeratorCancellation] CancellationToken cancellationToken =
default
    )
    {
        // Build your model's request object, specify that streaming is
requested
        MyModelRequest request = MyModelRequest.FromPrompt(prompt,
executionSettings);
        request.Stream = true;

        // Send the completion request via HTTP
        using var httpClient = new HttpClient();

        // Send a POST to your model with the serialized request in the
body
        using HttpResponseMessage httpResponse = await
httpClient.PostAsJsonAsync(
            ModelUrl,
            request,
            cancellationToken
);

        // Verify the request was completed successfully
        httpResponse.EnsureSuccessStatusCode();

        // Read your models response as a stream
        using StreamReader reader =
            new(await
httpResponse.Content.ReadAsStreamAsync(cancellationToken));

        // Iteratively read a chunk of the response until the end of
the stream
        // It is more efficient to use a buffer that is the same size
as the internal buffer of the stream
        // If the size of the internal buffer was unspecified when the
stream was constructed, its default size is 4 kilobytes (2048 UTF-16
characters)
        char[] buffer = new char[2048];
        while (!reader.EndOfStream)
        {
            // Check the cancellation token with each iteration
            cancellationToken.ThrowIfCancellationRequested();

            // Fill the buffer with the next set of characters, track
how many characters were read
            int readCount = reader.Read(buffer, 0, buffer.Length);
        }
    }
}
```

```

        // Convert the character buffer to a string, only include
        // as many characters as were just read
        string chunk = new(buffer, 0, readCount);

        yield return new StreamingTextContent(chunk);
    }
}

public async Task<IReadOnlyList<TextContent>> GetTextContentsAsync(
    string prompt,
    PromptExecutionSettings? executionSettings = null,
    Kernel? kernel = null,
    CancellationToken cancellationToken = default
)
{
    // Build your model's request object
    MyModelRequest request = MyModelRequest.FromPrompt(prompt,
executionSettings);

    // Send the completion request via HTTP
    using var httpClient = new HttpClient();

    // Send a POST to your model with the serialized request in the
    // body
    using HttpResponseMessage httpResponse = await
    httpClient.PostAsJsonAsync(
        ModelUrl,
        request,
        cancellationToken
    );

    // Verify the request was completed successfully
    httpResponse.EnsureSuccessStatusCode();

    // Deserialize the response body to your model's response
    // object
    // Handle when the deserialization fails and returns null
    MyModelResponse response =
        await
    httpResponse.Content.ReadFromJsonAsync<MyModelResponse>
    (cancellationToken)
    ?? throw new Exception("Failed to deserialize response from
model");

    // Convert your model's response into a list of
    ChatMessageContent
    return response
        .Completions.Select<string, TextContent>(completion =>
new(completion))
        .ToImmutableList();
}
}

```

2. Inclua a nova classe de serviço ao compilar o Kernel. Por exemplo:

```
C#  
  
IKernelBuilder builder = Kernel.CreateBuilder();  
  
// Add your text generation service as a singleton instance  
builder.Services.AddKeyedSingleton<ITextGenerationService>(  
    "myTextService1",  
    new MyTextGenerationService  
    {  
        // Specify any properties specific to your service, such as the  
        url or API key  
        ModelUrl = "https://localhost:38748",  
        ModelApiKey = "myApiKey"  
    }  
);  
  
// Alternatively, add your text generation service as a factory method  
builder.Services.AddKeyedSingleton<ITextGenerationService>(  
    "myTextService2",  
    (_, _) =>  
        new MyTextGenerationService  
        {  
            // Specify any properties specific to your service, such as  
            the url or API key  
            ModelUrl = "https://localhost:38748",  
            ModelApiKey = "myApiKey"  
        }  
);  
  
// Add any other Kernel services or configurations  
// ...  
Kernel kernel = builder.Build();
```

3. Envie um prompt de geração de texto para o modelo diretamente por meio do Kernel ou usando a classe de serviço. Por exemplo:

```
C#  
  
var executionSettings = new PromptExecutionSettings  
{  
    // Add execution settings, such as the ModelID and ExtensionData  
    ModelId = "MyModelId",  
    ExtensionData = new Dictionary<string, object> { { "MaxTokens", 500 } }  
};  
  
// Send a prompt to your model directly through the Kernel  
// The Kernel response will be null if the model can't be reached  
string prompt = "Please list three services offered by Azure";  
string? response = await kernel.InvokePromptAsync<string>(prompt);
```

```

Console.WriteLine($"Output: {response}");

// Alternatively, send a prompt to your model through the text
generation service
ITextGenerationService textService =
kernel.GetRequiredService<ITextGenerationService>();
TextContent responseContents = await textService.GetTextContentAsync(
    prompt,
    executionSettings
);
Console.WriteLine($"Output: {responseContents.Text}");

```

Implementar a conclusão de chat usando um modelo local

A seção a seguir mostra como você pode integrar seu modelo ao SDK do Kernel Semântico e usá-lo para conclusões de chat.

1. Crie uma classe de serviço que implemente a interface `IChatCompletionService`.

Por exemplo:

C#

```

class MyChatCompletionService : IChatCompletionService
{
    private IReadOnlyDictionary<string, object?>? _attributes;
    public IReadOnlyDictionary<string, object?> Attributes =>
        _attributes ??= new Dictionary<string, object?>();

    public string ModelUrl { get; init; } = "<default url to your
model's Chat API>";
    public required string ModelApiKey { get; init; }

    public async Task<IReadOnlyList<ChatMessageContent>>
GetChatMessageContentsAsync(
        ChatHistory chatHistory,
        PromptExecutionSettings? executionSettings = null,
        Kernel? kernel = null,
        CancellationToken cancellationToken = default
    )
    {
        // Build your model's request object
        MyModelRequest request =
MyModelRequest.FromChatHistory(chatHistory, executionSettings);

        // Send the completion request via HTTP
        using var httpClient = new HttpClient();

        // Send a POST to your model with the serialized request in the
body
    }
}

```

```
        using HttpResponseMessage httpResponse = await
httpClient.PostAsJsonAsync(
    ModelUrl,
    request,
    cancellationToken
);

    // Verify the request was completed successfully
    httpResponse.EnsureSuccessStatusCode();

    // Deserialize the response body to your model's response
    object
    // Handle when the deserialization fails and returns null
    MyModelResponse response =
        await
    httpResponse.Content.ReadFromJsonAsync<MyModelResponse>
(cancellationToken)
    ?? throw new Exception("Failed to deserialize response from
model");

    // Convert your model's response into a list of
    ChatMessageContent
    return response
        .Completions.Select<string, ChatMessageContent>(completion
=>
    new(AuthorRole.Assistant, completion)
)
    .ToImmutableList();
}

public async IAsyncEnumerable<StreamingChatMessageContent>
GetStreamingChatMessageContentsAsync(
    ChatHistory chatHistory,
    PromptExecutionSettings? executionSettings = null,
    Kernel? kernel = null,
    [EnumeratorCancellation] CancellationToken cancellationToken =
default
)
{
    // Build your model's request object, specify that streaming is
requested
    MyModelRequest request =
    MyModelRequest.FromChatHistory(chatHistory, executionSettings);
    request.Stream = true;

    // Send the completion request via HTTP
    using var httpClient = new HttpClient();

    // Send a POST to your model with the serialized request in the
body
    using HttpResponseMessage httpResponse = await
httpClient.PostAsJsonAsync(
    ModelUrl,
    request,
    cancellationToken
```

```

    );

    // Verify the request was completed successfully
    httpResponse.EnsureSuccessStatusCode();

    // Read your models response as a stream
    using StreamReader reader =
        new(await
    httpResponse.Content.ReadAsStreamAsync(cancellationToken));

    // Iteratively read a chunk of the response until the end of
    the stream
    // It is more efficient to use a buffer that is the same size
    as the internal buffer of the stream
    // If the size of the internal buffer was unspecified when the
    stream was constructed, its default size is 4 kilobytes (2048 UTF-16
    characters)
    char[] buffer = new char[2048];
    while (!reader.EndOfStream)
    {
        // Check the cancellation token with each iteration
        cancellationToken.ThrowIfCancellationRequested();

        // Fill the buffer with the next set of characters, track
        how many characters were read
        int readCount = reader.Read(buffer, 0, buffer.Length);

        // Convert the character buffer to a string, only include
        as many characters as were just read
        string chunk = new(buffer, 0, readCount);

        yield return new
StreamingChatMessageContent(AuthorRole.Assistant, chunk);
    }
}
}

```

2. Inclua a nova classe de serviço ao compilar o `Kernel`. Por exemplo:

```

C#

IKernelBuilder builder = Kernel.CreateBuilder();

// Add your chat completion service as a singleton instance
builder.Services.AddKeyedSingleton<IChatCompletionService>(
    "myChatService1",
    new MyChatCompletionService
    {
        // Specify any properties specific to your service, such as the
        url or API key
        ModelUrl = "https://localhost:38748",
        ModelApiKey = "myApiKey"
    }
}

```

```

);

// Alternatively, add your chat completion service as a factory method
builder.Services.AddKeyedSingleton<IChatCompletionService>(
    "myChatService2",
    (_, _) =>
        new MyChatCompletionService
    {
        // Specify any properties specific to your service, such as
        // the url or API key
        ModelUrl = "https://localhost:38748",
        ModelApiKey = "myApiKey"
    }
);

// Add any other Kernel services or configurations
// ...
Kernel kernel = builder.Build();

```

3. Envie um prompt de conclusão de chat para o modelo diretamente por meio do `Kernel` ou usando a classe de serviço. Por exemplo:

C#

```

var executionSettings = new PromptExecutionSettings
{
    // Add execution settings, such as the ModelID and ExtensionData
    ModelId = "MyModelId",
    ExtensionData = new Dictionary<string, object> { { "MaxTokens", 500 } }
};

// Send a string representation of the chat history to your model
// directly through the Kernel
// This uses a special syntax to denote the role for each message
// For more information on this syntax see:
// https://learn.microsoft.com/en-us/semantic-kernel/prompts/your-
// first-prompt?tabs=Csharp
string prompt = """
<message role="system">the initial system message for your chat
history</message>
<message role="user">the user's initial message</message>
""";

string? response = await kernel.InvokePromptAsync<string>(prompt);
Console.WriteLine($"Output: {response}");

// Alternatively, send a prompt to your model through the chat
// completion service
// First, initialize a chat history with your initial system message
string systemMessage = "<the initial system message for your chat
history>";
Console.WriteLine($"System Prompt: {systemMessage}");

```

```
var chatHistory = new ChatHistory(systemMessage);

// Add the user's input to your chat history
string userRequest = "<the user's initial message>";
Console.WriteLine($"User: {userRequest}");
chatHistory.AddUserMessage(userRequest);

// Get the models response and add it to the chat history
IChatCompletionService service =
kernel.GetRequiredService<IChatCompletionService>();
ChatMessageContent responseMessage = await
service.GetChatMessageContentAsync(
    chatHistory,
    executionSettings
);
Console.WriteLine($"Assistant: {responseMessage.Content}");
chatHistory.Add(responseMessage);

// Continue sending and receiving messages between the user and model
// ...
```

Conteúdo relacionado

- O que é Kernel Semântico
- Noções básicas sobre plug-ins de IA no Kernel Semântico

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).

.NET

Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 [Abrir um problema de documentação](#)

 [Fornecer comentários sobre o produto](#)

Trabalhar com a filtragem de conteúdo do OpenAI em um aplicativo do .NET

Artigo • 27/05/2024

Este artigo demonstra como lidar com preocupações de filtragem de conteúdo em um aplicativo do .NET. O Serviço OpenAI do Azure inclui um sistema de filtragem de conteúdo que funciona juntamente com os principais modelos. Esse sistema funciona executando o prompt e a conclusão por meio de um conjunto de modelos de classificação destinados a detectar e impedir a saída de conteúdo prejudicial. O sistema de filtragem de conteúdo detecta e executa ações em categorias específicas de conteúdo potencialmente prejudicial em prompts de entrada e conclusões de saída. As variações nas configurações de API e no design do aplicativo podem afetar os preenchimentos e, portanto, o comportamento de filtragem.

A documentação de [Filtragem de Conteúdo](#) fornece uma exploração mais detalhada dos conceitos e preocupações de filtragem de conteúdo. Este artigo fornece exemplos de como trabalhar com recursos de filtragem de conteúdo programaticamente em um aplicativo do .NET.

Pré-requisitos

- Uma conta do Azure com uma assinatura ativa. [Crie uma conta gratuitamente](#).
- [SDK .NET](#)
- [Criar e implantar um recurso do Serviço OpenAI do Azure](#)

Configurar e testar o filtro de conteúdo

Para usar o código de exemplo neste artigo, você precisa criar e atribuir um filtro de conteúdo ao seu modelo do OpenAI.

1. [Crie e atribua um filtro de conteúdo](#) ao modelo GPT-35 ou GPT-4 provisionado.
2. Adicione o pacote NuGet [Azure.AI.OpenAI](#) ao seu projeto.

CLI do .NET

```
dotnet add package Azure.AI.OpenAI
```

3. Crie um fluxo de conclusão de chat simples no seu aplicativo do .NET usando o `OpenAiClient`. Substitua os valores `YOUR_OPENAI_ENDPOINT`, `YOUR_OPENAI_KEY` e

YOUR_OPENAI_DEPLOYMENT pelos seus próprios valores.

C#

```
using Azure;
using Azure.AI.OpenAI;

string endpoint = "YOUR_OPENAI_ENDPOINT";
string key = "YOUR_OPENAI_KEY";

OpenAIClient client = new(new Uri(endpoint), new
AzureKeyCredential(key));

var chatCompletionsOptions = new ChatCompletionsOptions()
{
    DeploymentName = "YOUR_DEPLOYMENT_NAME",
    Messages =
    {
        new ChatRequestSystemMessage("You are a helpful assistant."),
        new ChatRequestUserMessage("YOUR_PROMPT")
    }
};

Response<ChatCompletions> response =
client.GetChatCompletions(chatCompletionsOptions);
Console.WriteLine(response.Value.Choices[0].Message.Content);
Console.WriteLine();
```

4. Imprima os resultados de filtragem de conteúdo para cada categoria.

C#

```
foreach (var promptFilterResult in response.Value.PromptFilterResults)
{
    var results = promptFilterResult.ContentFilterResults;
    Console.WriteLine($"Hate category is filtered:
{results.Hate.Filtered} with {results.Hate.Severity}
severity.");
    Console.WriteLine($"Self-harm category is filtered:
{results.SelfHarm.Filtered} with {results.SelfHarm.Severity}
severity.");
    Console.WriteLine($"Sexual category is filtered:
{results.Sexual.Filtered} with {results.Sexual.Severity}
severity.");
    Console.WriteLine($"Violence category is filtered:
{results.Violence.Filtered} with {results.Violence.Severity}
severity.");
}
```

5. Substitua o espaço reservado YOUR_PROMPT por sua própria mensagem e execute o aplicativo para experimentar os resultados da filtragem de conteúdo. A saída a

seguir mostra um exemplo de um prompt que dispara um resultado de filtragem de conteúdo de baixa severidade:

Saída

```
I am sorry if I have done anything to upset you.  
Is there anything I can do to assist you and make things better?  
  
Hate category is filtered: False with low severity.  
SelfHarm category is filtered: False with safe severity.  
Sexual category is filtered: False with safe severity.  
Violence category is filtered: False with low severity.
```

Conteúdo relacionado

- [Criar e atribuir um filtro de conteúdo](#)
- [Conceitos de Filtragem de Conteúdo](#)
- [Criar um aplicativo de chat](#)

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).

.NET

Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 [Abrir um problema de documentação](#)

 [Fornecer comentários sobre o produto](#)

Introdução à amostra de chats empresariais do .NET usando RAG

Artigo • 15/05/2024

Este artigo mostra como implantar e executar a [Amostra de aplicativo de chat empresarial para .NET](#). Esta amostra implementa um aplicativo de chat usando C#, Serviço OpenAI do Azure e a [Geração Aumentada de Recuperação \(RAG\)](#) na Pesquisa de IA do Azure para obter respostas sobre os benefícios dos funcionários em uma empresa fictícia. O aplicativo de chat de benefícios dos funcionários é semeado com arquivos PDF, incluindo um manual de funcionários, um documento de benefícios e uma lista de funções e expectativas da empresa.

- [Vídeo de demonstração](#)

Seguindo as instruções neste artigo, você vai:

- Implantar um aplicativo de chat no Azure.
- Obtenha respostas sobre os benefícios dos funcionários.
- Altere as configurações para alterar o comportamento das respostas.

Depois de concluir este procedimento, você poderá começar a modificar o novo projeto com seu código personalizado.

Este artigo faz parte de uma coleção de artigos que mostram como criar um aplicativo de chat usando o Serviço OpenAI do Azure e a Pesquisa de IA do Azure.

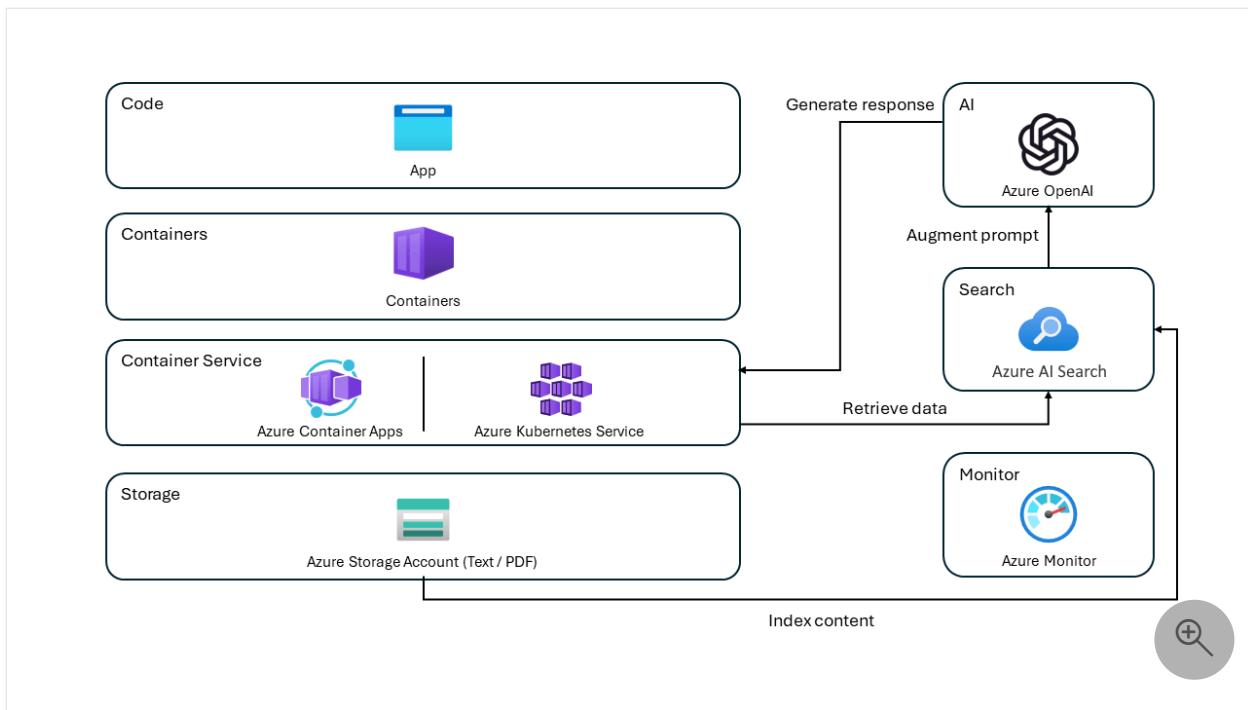
Outros artigos na coleção incluem:

- [Python](#)
- [JavaScript](#)
- [Java](#)

Visão geral da arquitetura

Neste aplicativo de exemplo, uma empresa fictícia chamada Contoso Electronics fornece a experiência do aplicativo de chat aos seus funcionários para fazer perguntas sobre benefícios, políticas internas, descrições de trabalho e funções.

A arquitetura do aplicativo de chat é mostrada no diagrama a seguir:



- **Interface do usuário** – a interface de chat do aplicativo é um aplicativo [Blazor WebAssembly](#). Esta interface é o que aceita consultas de usuário, roteia a solicitação para o back-end do aplicativo e exibe respostas geradas.
- **Back-end** – o back-end do aplicativo é uma [API mínima do ASP.NET Core](#). O back-end hospeda o aplicativo Web estático Blazor e é o que orquestra as interações entre os diferentes serviços. Os serviços usados neste aplicativo incluem:
 - [Azure Cognitive Search](#) – indexa documentos dos dados armazenados em uma conta do Armazenamento do Microsoft Azure. Isso torna os documentos pesquisáveis usando recursos de [busca em vetores](#).
 - [Serviço OpenAI do Azure](#) – fornece os modelos de linguagem grandes (LLMs) para gerar respostas. O [Kernel Semântico](#) é usado em conjunto com o Serviço OpenAI do Azure para orquestrar os fluxos de trabalho de IA mais complexos.

Custo

A maioria dos recursos nessa arquitetura usa um tipo de preço básico ou de consumo. O preço de consumo é baseado no uso, o que significa que você paga apenas pelo que usa. Para concluir este artigo, haverá uma cobrança, mas será mínima. Quando terminar de usar o artigo, você poderá excluir os recursos para parar de incorrer em encargos.

Para obter mais informações, consulte [Exemplos do Azure: Custo no repositório de exemplo ↗](#).

Pré-requisitos

Um ambiente de [contêiner de desenvolvimento](#) está disponível com todas as dependências necessárias para concluir este artigo. Você pode executar o contêiner de desenvolvimento em Codespaces do GitHub (em um navegador) ou localmente usando o Visual Studio Code.

Para acompanhar este artigo, você precisa dos seguintes pré-requisitos:

Codespaces (recomendado)

1. Uma assinatura do Azure – [crie uma gratuitamente](#)
2. Permissões de conta do Azure – sua conta do Azure deve ter permissões Microsoft.Authorization/roleAssignments/write, como [Administrador de Acesso do Usuário](#) ou [Proprietário](#).
3. Acesso permitido ao OpenAI do Azure na assinatura do Azure desejada. No momento, o acesso a esse serviço é permitido somente por aplicativo. Você pode solicitar acesso ao Serviço OpenAI do Azure preenchendo o formulário em <https://aka.ms/oai/access>. Abra um problema neste repositório para entrar em contato conosco se você tiver algum problema.
4. GitHub

Abrir o ambiente de desenvolvimento

Comece agora com um ambiente de desenvolvimento que tenha todas as dependências instaladas para concluir este artigo.

Codespaces do GitHub (recomendado)

O [GitHub Codespaces](#) executa um contêiner de desenvolvimento gerenciado pelo GitHub com o [Visual Studio Code para Web](#) como interface do usuário. Para o ambiente de desenvolvimento mais simples, use os Codespaces do GitHub para que você tenha as ferramentas e dependências de desenvolvedor corretas pré-instaladas para concluir este artigo.

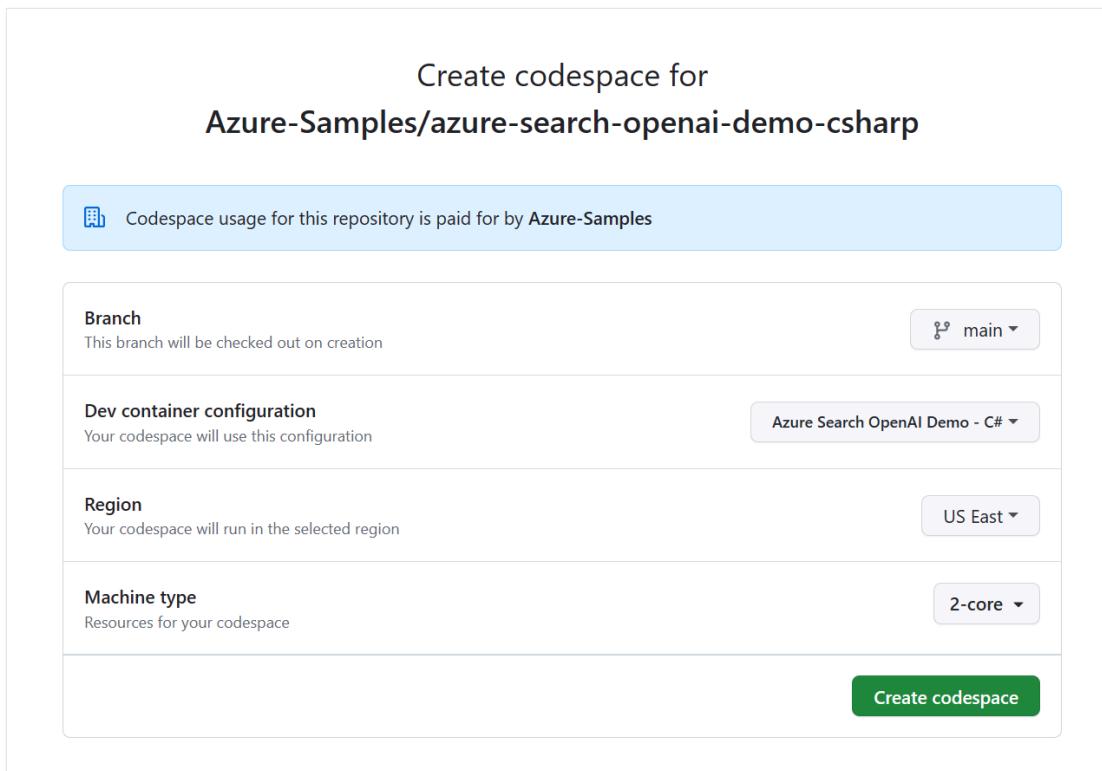
ⓘ Importante

Todas as contas do GitHub podem usar Codespaces por até 60 horas gratuitas por mês com 2 instâncias principais. Para saber mais, confira [Armazenamento e horas por núcleo incluídos mensalmente no GitHub Codespaces](#).

1. Inicie o processo para criar um GitHub Codespace no branch `main` do repositório GitHub [Azure-Samples/azure-search-openai-demo-csharp](#).
2. Clique com o botão direito do mouse no botão a seguir e selecione *Abrir link em novas janelas* para ter o ambiente de desenvolvimento e a documentação disponíveis ao mesmo tempo.

[Abra este projeto no Codespaces do GitHub](#)

3. Na página **Criar codespace**, examine as configurações do codespace e selecione **Criar novo codespace**:



4. Aguarde até que o codespace seja iniciado. Esse processo de inicialização pode levar alguns minutos.
5. No terminal na parte inferior da tela, entre no Azure com o Azure Developer CLI.

```
Bash
azd auth login
```

6. Copie o código do terminal e cole-o em um navegador. Siga as instruções para autenticar com sua conta do Azure.

7. As tarefas restantes neste artigo ocorrem no contexto desse contêiner de desenvolvimento.

Implantar e executar

O repositório de exemplo contém todos os arquivos de código e configuração necessários para implantar um aplicativo de chat no Azure. As etapas a seguir explicam o processo de implantação do exemplo no Azure.

Implantar aplicativo de chat no Azure

Importante

Os recursos do Azure criados nesta seção geram custos imediatos, principalmente do recurso de Pesquisa de IA do Azure. Esses recursos podem acumular custos mesmo se você interromper o comando antes que ele seja totalmente executado.

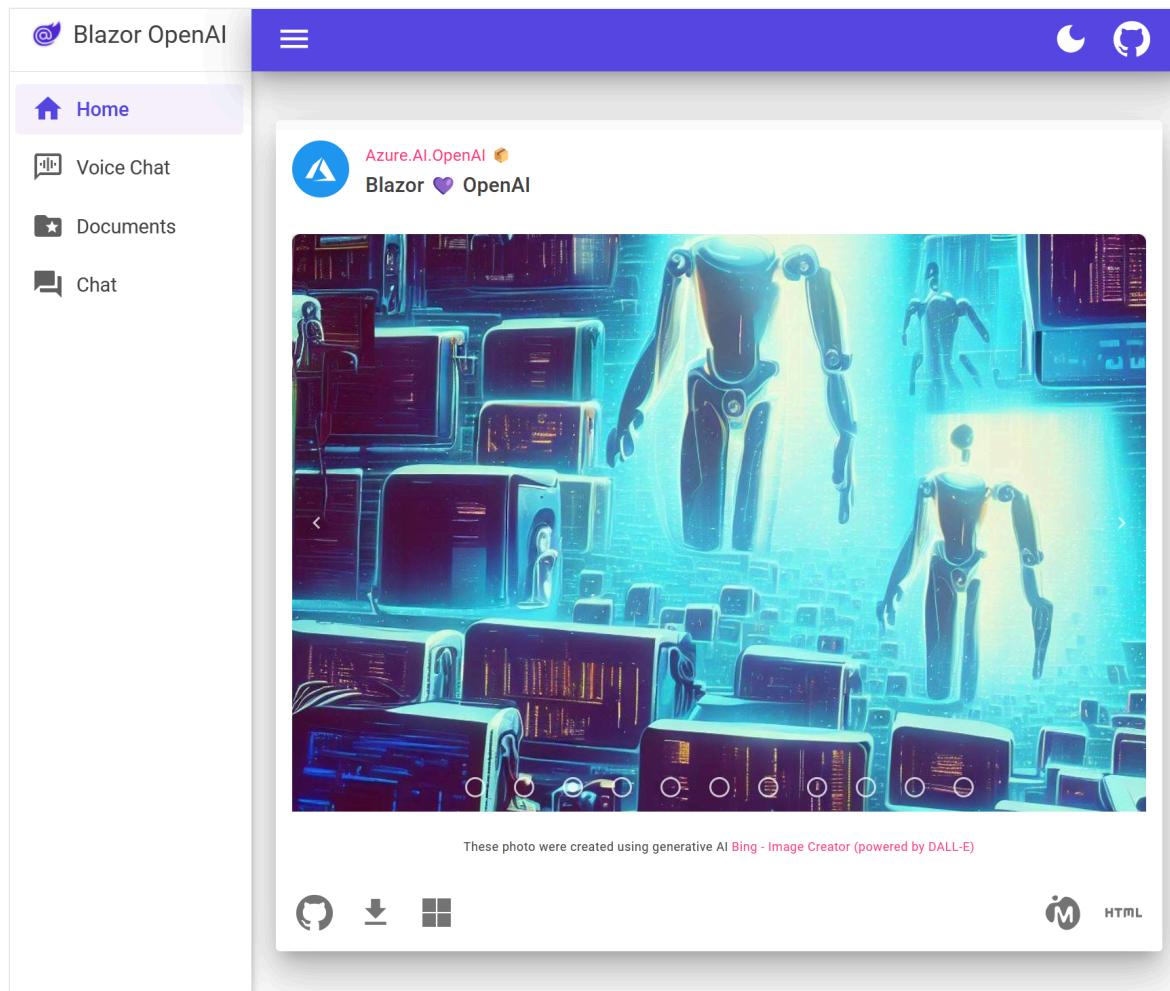
1. Execute o seguinte comando do Azure Developer CLI para provisionar os recursos do Azure e implantar o código-fonte:

```
Bash
```

```
azd up
```

2. Quando solicitado que você insira um nome de ambiente, mantenha-o curto e minúsculo. Por exemplo, `myenv`. É usado como parte do nome do grupo de recursos.
3. Quando solicitado, selecione uma assinatura para criar os recursos.
4. Quando solicitado que você selecione um local na primeira vez, selecione um local próximo a você. Esse local é usado para a maioria dos recursos, incluindo hospedagem.
5. Se for solicitado que você solicite um local para o modelo OpenAI, selecione um local próximo a você. Se o mesmo local estiver disponível como seu primeiro local, selecione-o.
6. Aguarde até que o aplicativo seja implantado. Pode levar até 20 minutos para que a implantação seja concluída.

7. Depois que o aplicativo tiver sido implantado com êxito, você verá uma URL exibida no terminal.
8. Selecione essa URL rotulada [Deploying service web](#) para abrir o aplicativo de chat em um navegador.



Usar o aplicativo de chat para obter respostas de arquivos PDF

O aplicativo de chat é pré-carregado com informações de benefícios dos funcionários de [arquivos PDF](#). Você pode usar o aplicativo de chat para fazer perguntas sobre os benefícios. As etapas a seguir explicam o processo de uso do aplicativo de chat.

1. No navegador, navegue até a página **Chat** usando a navegação à esquerda.
2. Selecione ou insira "O que está incluído no meu plano Northwind Health Plus que não está no padrão?" na caixa de texto do chat. Sua resposta é *semelhante* à imagem a seguir.

What is included in my Northwind Health Plus plan that is not in standard?
Asked at 3:06:54 PM on 5/10/24

ANSWER **THOUGHT PROCESS** **SUPPORTING CONTENT**

Specialty care services such as physical therapy, occupational therapy, and mental health services are included in the Northwind Health Plus plan but not in the standard plan. 1

Citations:

1. Source: Northwind_Health_Plus_Benefits_Details-47.pdf

Follow-up questions:

What conditions are covered under mental health services?

Is there a limit on the number of therapy sessions allowed per year?

Are specialty care services subject to a separate deductible?

Prompt
Enter OpenAI + Azure Search prompt
Use Shift + Enter for new lines.
0 / 1000

CHAT 

3. Na resposta, selecione uma citação. Uma janela pop-up será aberta exibindo a origem das informações.

Northwind_Health_Plus_Benefits_Details-90.pdf

Tips for Employees:

1. Read your Summary Plan Description (SPD) carefully to understand the benefits available to you under Northwind Health Plus.
2. Familiarize yourself with the applicable laws and regulations, such as ERISA, the Affordable Care Act (ACA), and the Mental Health Parity and Addiction Equity Act (MHPAEA).
3. Be aware of the coverage and limits your plan provides.
4. Be aware of any exclusions or exceptions that may apply to your plan.
5. If you feel you have been discriminated against, contact the Department of Labor.

By understanding the applicable laws and regulations and the coverage and limits of your plan, you can ensure that you are getting the most out of your Northwind Health Plus benefits.

Entire Contract
OTHER INFORMATION ABOUT THIS PLAN - Entire Contract

The Northwind Health Plus plan is a contract between you and Northwind Health. It is important to understand that this document contains the entire contract. This contract includes the plan documents that you receive from Northwind Health, the Northwind Health Plus plan summary, and any additional contracts or documents that you may have received from Northwind Health.

It is important to remember that any changes made to this plan must be in writing and signed by both you and Northwind Health. Additionally, if something in the plan is not included in the plan documents or summary, then it does not apply to the plan.

You should also be aware that the Northwind Health Plus plan may contain certain exceptions, exclusions, and limitations. It is important to familiarize yourself with the plan documents to make sure that you understand what services are covered and which are not



4. Navegue entre as guias na parte superior da caixa de resposta para entender como a resposta foi gerada.

 Expandir a tabela

Tab	Descrição
Processo de pensamento	Esse é um script das interações no chat. Você pode exibir o prompt do sistema (<code>content</code>) e a pergunta do usuário (<code>content</code>).
Conteúdo de suporte	Isso inclui as informações para responder à sua pergunta e ao material de origem. O número de citações de material de origem é observado nas Configurações do Desenvolvedor . O valor padrão é 3.
Citação	Isso exibe a página de origem que contém a citação.

5. Quando terminar, navegue de volta para a guia de resposta.

Usar as configurações do aplicativo de chat para alterar o comportamento das respostas

A inteligência do chat é determinada pelo modelo OpenAI e pelas configurações usadas para interagir com o modelo.

Configure Answer Generation

Override prompt template

Override prompt template

Retrieve this many documents from search

3

Exclude category

Exclude category

Use semantic ranker for retrieval

Retrieval Mode

Text Hybrid Vector

Use query-contextual summaries
instead of whole documents

Suggest follow-up questions

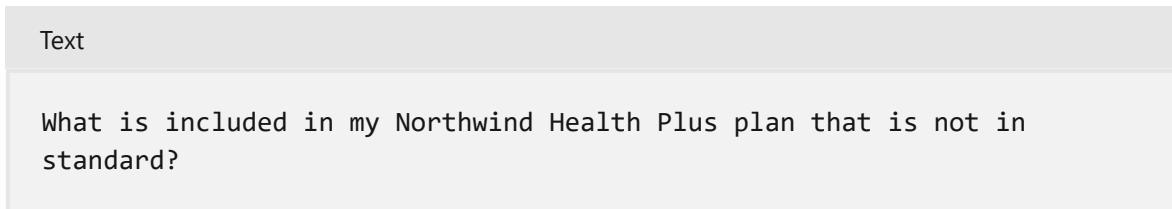
 CLOSE

 Expandir a tabela

Configuração	Descrição
Substituir modelo de prompt	Esse é o prompt usado para gerar a resposta.
Recuperar muitos resultados da pesquisa	Esse é o número de resultados da pesquisa que são usados para gerar a resposta. Você pode ver essas fontes retornadas nas guias <i>Processo de pensamento</i> e <i>Conteúdo de suporte</i> da citação.
Excluir categoria	Essa é a categoria de documentos que são excluídos dos resultados da pesquisa.
Usar o classificador semântico para recuperação	Esse é um recurso da Pesquisa de IA do Azure que usa o aprendizado de máquina para melhorar a relevância dos resultados da pesquisa.
Modo de recuperação	Vetores + Texto significa que os resultados da pesquisa são baseados no texto dos documentos e nas inserções dos documentos. Vetores significa que os resultados da pesquisa são baseados nas inserções dos documentos. Texto significa que os resultados da pesquisa são baseados no texto dos documentos.
Usar resumos contextuais de consulta em vez de documentos inteiros	Quando ambos <code>Use semantic ranker</code> e <code>Use query-contextual summaries</code> são verificados, o LLM usa legendas extraídas de passagens-chave, em vez de todas as passagens, nos documentos mais bem classificados.
Sugerir perguntas de acompanhamento	Faça com que o aplicativo de chat sugira perguntas de acompanhamento com base na resposta.

As etapas a seguir explicam o processo de alteração das configurações.

1. No navegador, selecione o ícone de engrenagem no canto superior direito da página.
2. Se não estiver selecionado, marque a caixa de seleção **Sugerir perguntas de acompanhamento** e faça a mesma pergunta novamente.



O chat pode retornar com sugestões de perguntas de acompanhamento.

3. Na guia **Configurações**, desmarque **Usar classificador semântico para recuperação**.
4. Faça a mesma pergunta novamente.

Text

What is my deductible?

5. Qual é a diferença nas respostas?

A resposta que utilizou o classificador semântico forneceu uma única resposta. A resposta sem classificação semântica retornou uma resposta menos direta.

Limpar os recursos

Limpar recursos do Azure

Os recursos do Azure criados neste artigo são cobrados para sua assinatura do Azure. Se você não espera precisar desses recursos no futuro, exclua-os para evitar incorrer em mais encargos.

Execute o seguinte comando do Azure Developer CLI para excluir os recursos do Azure e remover o código-fonte:

Bash

```
azd down --purge
```

Limpar GitHub Codespaces

Codespaces do GitHub

A exclusão do ambiente GitHub Codespaces garante que você possa maximizar a quantidade de horas gratuitas por núcleo que você tem direito na sua conta.

ⓘ Importante

Para saber mais sobre os direitos da sua conta do GitHub, confira [O GitHub Codespaces inclui mensalmente armazenamento e horas de núcleo](#).

1. Entre no painel do GitHub Codespaces (<https://github.com/codespaces>).
2. Localize os codespaces atualmente em execução provenientes do repositório GitHub [Azure-Samples/azure-search-openai-demo-csharp](#).

The screenshot shows the GitHub Codespaces interface. At the top, there's a search bar and various navigation icons. Below that, a sidebar on the left lists 'All' (1) and 'Templates'. Under 'By repository', it shows 'Azure-Samples/azure-search-openai-demo' (1). The main area is titled 'Your codespaces' and features a section for 'Explore quick start templates' with options for 'Blank', 'React', and 'Jupyter Notebook'. Below this, a box labeled 'Owned by developer-bob' contains the details for the 'effective orbit' codespace. This box is highlighted with a red border. Inside, it shows the repository 'Azure-Samples/azure-search-openai-demo', the branch 'main', and status 'No changes'. To the right, it lists the machine configuration as '2-core + 8GB RAM + 32GB', the status 'Retrieving...', and the last usage time '7 minutes ago'. There are also three dots (...).

3. Abra o menu de contexto do codespace e selecione Excluir.

This screenshot is similar to the one above, showing the 'effective orbit' codespace in the 'Owned by developer-bob' section. However, a context menu is now open over the codespace card. The menu items include 'Open in ...', 'Rename', 'Export changes to a fork', 'Change machine type', 'Keep codespace', and 'Delete'. The 'Delete' option is highlighted with a red box.

Obter ajuda

Este repositório de exemplo oferece [informações de solução de problemas ↗](#).

Se o problema não for resolvido, registre seu problema nos problemas do repositório.

Próximas etapas

- [Repositório GitHub do aplicativo de chat empresarial ↗](#)

- Criar um aplicativo de chat com a arquitetura de solução de práticas recomendadas do Azure OpenAI ↗
- Controle de acesso em aplicativos de IA generativos com a Pesquisa de IA do Azure ↗
- Criar uma solução OpenAI pronta para empresas com o Gerenciamento de API do Azure ↗
- Superando a busca em vetores com recursos de recuperação e classificação híbridas ↗

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).

.NET

Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

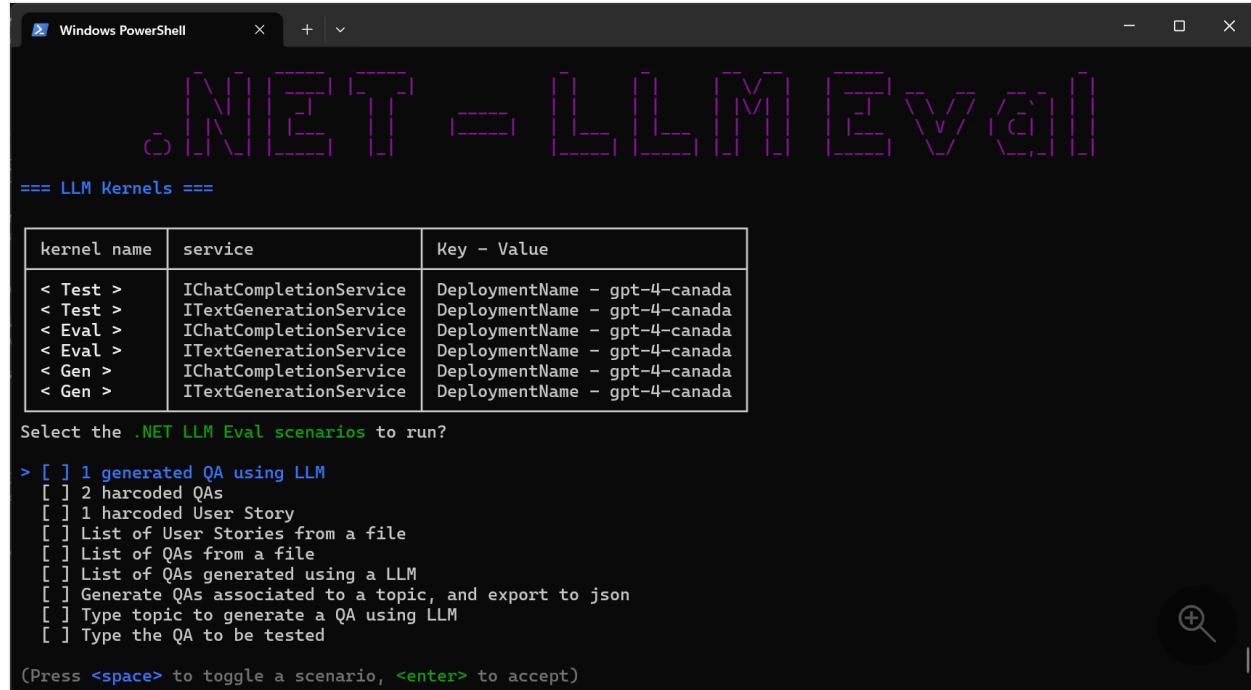
 Abrir um problema de documentação

 Fornecer comentários sobre o produto

Tutorial: Evaluate an LLM's prompt completions

Article • 11/24/2024

In this tutorial, you evaluate the coherence, relevance, and groundedness of an LLM's prompt completions using Azure OpenAI and the Semantic Kernel SDK for .NET.



In this tutorial, you learn how to:

- ✓ Clone and build the evaluation application
- ✓ Configure the models
- ✓ Generate evaluation test data
- ✓ Perform an evaluation of your LLM
- ✓ Review the results of an evaluation

If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

- [.NET 8.0 SDK](#)
- [Create and deploy a GPT-4 model to Azure OpenAI Service](#)

1 - Clone the evaluation application

Get the source for the evaluation application and ensure it can be built.

1. Clone the repository [dotnet/ai-samples](#).
2. From a terminal or command prompt, navigate to the `ai-samples/src/llm-eval` directory.
3. Build the evaluation application:

```
.NET CLI  
dotnet build .
```

2 - Configure the models

Set the model to be tested and the models to perform evaluations and generate test data.

It's best to use a GPT-4 model for performing evaluation. You can use an Azure OpenAI resource, an OpenAI instance, or any LLM supported by the Semantic Kernel SDK. This article uses a GPT-4 model deployed to an Azure OpenAI resource for evaluations.

The `KernelFactory` class (`src/LLMEval.Test/KernelFactory.cs`) creates the kernels for evaluations, generating test data, and the LLM being tested.

Configure the model to test

The evaluation application tests the model that the `KernelFactory.CreateKernelTest` method returns.

The Semantic Kernel SDK can integrate any model that supports the *OpenAI Chat Completion API*.

Update the `KernelFactory.CreateKernelTest` method to return a `Kernel` object that uses the model to be tested. For example, the following example creates a `Kernel` object that uses a Llama 3 model deployed and hosted locally using Ollama:

```
C#  
  
public static Kernel CreateKernelTest()  
{  
    IKernelBuilder builder = Kernel.CreateBuilder();  
  
    builder.AddOpenAIChatCompletion(  
        modelId: "phi3",  
        endpoint: new Uri("http://localhost:11434"),  
        apiKey: "api"
```

```
    );  
  
    return builder.Build();  
}
```

Configure the model to perform evaluations

Use .NET user secrets to store the Azure OpenAI deployment info. From the `ai-samples/src/l1m-eval/LLMEval.Test` directory, run the following commands:

.NET CLI

```
dotnet user-secrets init  
dotnet user-secrets set "AZURE_OPENAI_MODEL" "<deployment-name>"  
dotnet user-secrets set "AZURE_OPENAI_ENDPOINT" "<deployment-endpoint>"  
dotnet user-secrets set "AZURE_OPENAI_KEY" "<deployment-key>"
```

The evaluation application is configured to use these secrets to connect to an Azure OpenAI model to perform evaluations. You can update this configuration in the `KernelFactory.CreateKernelEval` method:

C#

```
public static Kernel CreateKernelEval()  
{  
    IConfigurationRoot config = new  
    ConfigurationBuilder().AddUserSecrets<Program>().Build();  
    IKernelBuilder builder = Kernel.CreateBuilder();  
  
    builder.AddAzureOpenAIChatCompletion(  
        config["AZURE_OPENAI_MODEL"],  
        config["AZURE_OPENAI_ENDPOINT"],  
        config["AZURE_OPENAI_KEY"]  
    );  
  
    return builder.Build();  
}
```

Configure the model to generate test data

The evaluation application is configured to use [the secrets set in the previous step](#) to connect to an Azure OpenAI model to generate test data. You can update this configuration in the `KernelFactory.CreateKernelGenerateData` method:

C#

```

public static Kernel CreateKernelGenerateData()
{
    IConfigurationRoot config = new ConfigurationBuilder().AddUserSecrets<Program>().Build();
    IKernelBuilder builder = Kernel.CreateBuilder();

    builder.AddAzureOpenAIChatCompletion(
        config["AZURE_OPENAI_MODEL"],
        config["AZURE_OPENAI_ENDPOINT"],
        config["AZURE_OPENAI_KEY"]
    );

    return builder.Build();
}

```

3 - Generate test data

The evaluation application compares an LLM's output to "ground truth" answers, which are ideal question-answer pairs. It's recommended to have at least 200 question-answer pairs for an evaluation.

You can use the evaluation application to generate an initial set of question-answer pairs. Then manually curate them by rewriting or removing any subpar answers.

Tips for generating test data:

- Generate more question-answer pairs than you need, then manually prune them based on quality and overlap. Remove low quality answers, and remove questions that are too similar to other questions.
- Be aware of the knowledge distribution so you effectively sample questions across the relevant knowledge space.
- Once your application is live, continually sample real user questions (within accordance to your privacy policy) to make sure you're representing the kinds of questions that users are asking.

1. From the `ai-samples/src/llm-eval/LLMEval.Test` directory, run the following command:

.NET CLI

`dotnet run .`

2. Select **Generate QAs associated to a topic, and export to json**, then press `Enter`.

```
Windows PowerShell x + - x
.NET Full LLM Env[]

==== LLM Kernels ====


| kernel name | service                | Key - Value                   |
|-------------|------------------------|-------------------------------|
| < Test >    | IChatCompletionService | DeploymentName - gpt-4-canada |
| < Test >    | ITextGenerationService | DeploymentName - gpt-4-canada |
| < Eval >    | IChatCompletionService | DeploymentName - gpt-4-canada |
| < Eval >    | ITextGenerationService | DeploymentName - gpt-4-canada |
| < Gen >     | IChatCompletionService | DeploymentName - gpt-4-canada |
| < Gen >     | ITextGenerationService | DeploymentName - gpt-4-canada |


Select the .NET LLM Eval scenarios to run?
[ ] 1 generated QA using LLM
[ ] 2 hardcoded QAs
[ ] 1 hardcoded User Story
[ ] List of User Stories from a file
[ ] List of QAs from a file
[ ] List of QAs generated using a LLM
> [X] Generate QAs associated to a topic, and export to json
[ ] Type topic to generate a QA using LLM
[ ] Type the QA to be tested
(Press <space> to toggle a scenario, <enter> to accept)
```

3. Enter the number of question-answer pairs to be generated and their topic.

```
Windows PowerShell x + - x
.NET Full LLM Env[]

==== LLM Kernels ====


| kernel name | service                | Key - Value                   |
|-------------|------------------------|-------------------------------|
| < Test >    | IChatCompletionService | DeploymentName - gpt-4-canada |
| < Test >    | ITextGenerationService | DeploymentName - gpt-4-canada |
| < Eval >    | IChatCompletionService | DeploymentName - gpt-4-canada |
| < Eval >    | ITextGenerationService | DeploymentName - gpt-4-canada |
| < Gen >     | IChatCompletionService | DeploymentName - gpt-4-canada |
| < Gen >     | ITextGenerationService | DeploymentName - gpt-4-canada |


==== Processing user selection ====
==== Generate a collection of QAs, for a specific topic ====
How many QAs do you want to generate? 200
Type the topic to generate the QAs Math
```

4. A preview of the generated question-answer pairs in JSON format is shown; enter the path of the file to save the JSON to.

```
{  
    "Question": "What is the square root of 81?",  
    "Answer": "9",  
    "Topic": "Math"  
},  
,  
    "Question": "What is the Pythagorean Theorem?",  
    "Answer": "In mathematics, the Pythagorean theorem, also known as Pythagoras\u2019s theorem, is a fundamental relation in Euclidean geometry among the three sides of a right triangle. It states that the square of the length of the hypotenuse (the side opposite the right angle) is equal to the sum of the squares of the lengths of the other two sides. This can be written as: a\u00b2 + b\u00b2 = c\u00b2.",  
    "Topic": "Math"  
},  
,  
    "Question": "What is the square root of 144?",  
    "Answer": "12",  
    "Topic": "Math"  
},  
,  
    "Question": "What is the square root of 16?",  
    "Answer": "4",  
    "Topic": "Math"  
},  
,  
    "Question": "What is the square root of 16?",  
    "Answer": "4",  
    "Topic": "Math"  
}  
]  
Type the file name to export the QAs. eval.json
```

5. Review the output JSON, and update or remove any incorrect or subpar answers.

4 - Perform an evaluation

Once you've curated the question-answer pairs, the evaluation application can use them to evaluate the outputs of the test model.

1. Copy the JSON file containing the question-answer pairs to `ai-samples/src/l1m-eval/LLMEval.Test/assets/qa-02.json`.
2. From the `ai-samples/src/l1m-eval/LLMEval.Test` directory, run the following command:

```
.NET CLI
```

```
dotnet run .
```

3. Select **List of QAs from a file**, then press `Enter`.

```

Windows PowerShell
+ - x
.NET LLM ENVIRONMENT
==== LLM Kernels ====


| kernel name | service                | Key - Value                   |
|-------------|------------------------|-------------------------------|
| < Test >    | IChatCompletionService | DeploymentName - gpt-4-canada |
| < Test >    | ITextGenerationService | DeploymentName - gpt-4-canada |
| < Eval >    | IChatCompletionService | DeploymentName - gpt-4-canada |
| < Eval >    | ITextGenerationService | DeploymentName - gpt-4-canada |
| < Gen >     | IChatCompletionService | DeploymentName - gpt-4-canada |
| < Gen >     | ITextGenerationService | DeploymentName - gpt-4-canada |


Select the .NET LLM Eval scenarios to run?
[ ] 1 generated QA using LLM
[ ] 2 hardcoded QAs
[ ] 1 hardcoded User Story
[ ] List of User Stories from a file
> [X] List of QAs from a file
[ ] List of QAs generated using a LLM
[ ] Generate QAs associated to a topic, and export to json
[ ] Type topic to generate a QA using LLM
[ ] Type the QA to be tested
(Press <space> to toggle a scenario, <enter> to accept)

```

4. The evaluation results are printed in a table format.

is a fundamental relation in Euclidean geometry among the three sides of a right triangle. It states that the square of the length of the hypotenuse (the side opposite the right angle) is equal to the sum of the squares of the lengths of the other two sides. This can be written as: $a^2 + b^2 = c^2$.", for this topic: "Math" The model was asked this question: "What is the square root of 144?", expecting this answer: "12", for this topic: "Math" The model was asked this question: "What is the square root of 16?", expecting this answer: "4", for this topic: "Math" The model was asked this question: "What is the square root of 16?", expecting this answer: "4", for this topic: "Math"	square of the length of the hypotenuse (the side opposite the right angle) is equal to the sum of the squares of the lengths of the other two sides. This can be written as: $a^2 + b^2 = c^2$. The model answer is: "12"	5	1	5	25
	The model answer is: "4"	4	1	5	24
	The model answer is: "4"	5	1	5	24

```

==== Complete. ====
PS C:\Source\ai-samples\src\llm-eval\LLMEval.Test>

```

5 - Review the evaluation results

The evaluation results [generated in the previous step](#) include a *coherence*, *relevance*, and *groundedness* metric. These metrics are similar to the built-in metrics provided by the Azure AI Studio.

- *Coherence*: Measures how well the language model can produce outputs that flow smoothly, read naturally, and resemble human-like language.
 - Based on `ai-samples/src/LLMEval.Core/_prompts/coherence/skprompt.txt`
- *Relevance*: Assesses the ability of answers to capture the key points of the context.
 - Based on `ai-samples/src/LLMEval.Core/_prompts/relevance/skprompt.txt`

- *Groundedness*: Assesses the correspondence between claims in an AI-generated answer and the source context, making sure that these claims are substantiated by the context.
 - Based on `ai-samples/src/LLMEval.Core/_prompts/groundedness/skprompt.txt`

Clean up resources

If you no longer need them, delete the Azure OpenAI resource and GPT-4 model deployment.

1. In the [Azure Portal](#), navigate to the Azure OpenAI resource.
2. Select the Azure OpenAI resource then select **Delete**.

Related content

- [Evaluation of generative AI applications](#)
- [What is Semantic Kernel?](#)
- [What is Azure OpenAI Service?](#)

Implement Azure OpenAI with RAG using vector search in a .NET app

Article • 11/24/2024

This tutorial explores integration of the RAG pattern using Open AI models and vector search capabilities in a .NET app. The sample application performs vector searches on custom data stored in Azure Cosmos DB for MongoDB and further refines the responses using generative AI models, such as GPT-35 and GPT-4. In the sections that follow, you'll set up a sample application and explore key code examples that demonstrate these concepts.

Prerequisites

- [.NET 8.0 installed](#)
- An [Azure Account](#)
- An [Azure Cosmos DB for MongoDB vCore](#) service
- An [Azure Open AI](#) service
 - Deploy `text-embedding-ada-002` model for embeddings
 - Deploy `gpt-35-turbo` model for chat completions

App overview

The Cosmos Recipe Guide app allows you to perform vector and AI driven searches against a set of recipe data. You can search directly for available recipes or prompt the app with ingredient names to find related recipes. The app and the sections ahead guide you through the following workflow to demonstrate this type of functionality:

1. Upload sample data to an Azure Cosmos DB for MongoDB database.
2. Create embeddings and a vector index for the uploaded sample data using the Azure OpenAI `text-embedding-ada-002` model.
3. Perform vector similarity search based on the user prompts.
4. Use the Azure OpenAI `gpt-35-turbo` completions model to compose more meaningful answers based on the search results data.

```
C:\ai-dotnet\AzureDataRetrie > CosmosDB RecipeApp
We have 13 vectorized recipe(s) and 0 non vectorized recipe(s).
Select an option to continue
> 1. Upload recipe(s) to Cosmos DB
2. Vectorize the recipe(s) and store it in Cosmos DB
3. Ask AI Assistant (search for a recipe by name or description, or ask a question)
4. Exit this Application
```

Get started

1. Clone the following GitHub repository:

Bash

```
git clone
https://github.com/microsoft/AzureDataRetrievalAugmentedGenerationSamples.git
```

2. In the *C#/CosmosDB-MongoDBvCore* folder, open the **CosmosRecipeGuide.sln** file.
3. In the *appsettings.json* file, replace the following config values with your Azure OpenAI and Azure CosmosDB for MongoDB values:

JSON

```
"OpenAIEndpoint": "https://<your-service-name>.openai.azure.com/",
"OpenAIKey": "<your-api-key>",
"OpenAIEmbeddingDeployment": "<your-ada-deployment-name>",
"OpenAIcompletionsDeployment": "<your-gpt-deployment-name>",
"MongoVcoreConnection": "<your-mongo-connection-string>"
```

4. Launch the app by pressing the **Start** button at the top of Visual Studio.

Explore the app

When you run the app for the first time, it connects to Azure Cosmos DB and reports that there are no recipes available yet. Follow the steps displayed by the app to begin

the core workflow.

1. Select **Upload recipe(s) to Cosmos DB** and press **Enter**. This command reads sample JSON files from the local project and uploads them to the Cosmos DB account.

The code from the *Utility.cs* class parses the local JSON files.

```
C#  
  
public static List<Recipe> ParseDocuments(string Folderpath)  
{  
    List<Recipe> recipes = new List<Recipe>();  
  
    Directory.GetFiles(Folderpath)  
        .ToList()  
        .ForEach(f =>  
    {  
        var jsonString= System.IO.File.ReadAllText(f);  
        Recipe recipe = JsonConvert.DeserializeObject<Recipe>(jsonString);  
        recipe.id = recipe.name.ToLower().Replace(" ", "");  
        ret.Add(recipe);  
    }  
);  
  
    return recipes;  
}
```

The `UpsertVectorAsync` method in the *VCoreMongoService.cs* file uploads the documents to Azure Cosmos DB for MongoDB.

```
C#  
  
public async Task UpsertVectorAsync(Recipe recipe)  
{  
    BsonDocument document = recipe.ToBsonDocument();  
  
    if (!document.Contains("_id"))  
    {  
        Console.WriteLine("UpsertVectorAsync: Document does not  
contain _id.");  
        throw new ArgumentException("UpsertVectorAsync: Document  
does not contain _id.");  
    }  
  
    string? _idValue = document["_id"].ToString();  
  
    try  
    {  
        var filter = Builders<BsonDocument>.Filter.Eq("_id",
```

```

        _idValue);
        var options = new ReplaceOptions { IsUpsert = true };
        await _recipeCollection.ReplaceOneAsync(filter, document,
options);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Exception: UpsertVectorAsync():
{ex.Message}");
        throw;
    }
}

```

2. Select Vectorize the recipe(s) and store them in Cosmos DB.

The JSON items uploaded to Cosmos DB do not contain embeddings and therefore are not optimized for RAG via vector search. An embedding is an information-dense, numerical representation of the semantic meaning of a piece of text. Vector searches are able to find items with contextually similar embeddings.

The `GetEmbeddingsAsync` method in the `OpenAIService.cs` file creates an embedding for each item in the database.

C#

```

public async Task<float[]?> GetEmbeddingsAsync(dynamic data)
{
    try
    {
        EmbeddingsOptions options = new EmbeddingsOptions(data)
        {
            Input = data
        };

        var response = await
        _openAIClient.GetEmbeddingsAsync(openAIEmbeddingDeployment, options);

        Embeddings embeddings = response.Value;
        float[] embedding = embeddings.Data[0].Embedding.ToArray();

        return embedding;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"GetEmbeddingsAsync Exception:
{ex.Message}");
        return null;
    }
}

```

The `CreateVectorIndexIfNotExists` in the `VCoreMongoService.cs` file creates a vector index, which enables you to perform vector similarity searches.

C#

```
public void CreateVectorIndexIfNotExists(string vectorIndexName)
{
    try
    {
        //Find if vector index exists in vectors collection
        using (IAsyncCursor<BsonDocument> indexCursor =
_recipeCollection.Indexes.List())
        {
            bool vectorIndexExists = indexCursor.ToList().Any(x =>
x["name"] == vectorIndexName);
            if (!vectorIndexExists)
            {
                BsonDocumentCommand<BsonDocument> command = new
BsonDocumentCommand<BsonDocument>(
                    BsonDocument.Parse(@"
                    { createIndexes: 'Recipe',
                      indexes: [{ 
                        name: 'vectorSearchIndex',
                        key: { embedding: 'cosmosSearch' },
                        cosmosSearchOptions: {
                            kind: 'vector-ivf',
                            numLists: 5,
                            similarity: 'COS',
                            dimensions: 1536 
                        }
                      }]
                    }"));
                BsonDocument result = _database.RunCommand(command);
                if (result["ok"] != 1)
                {
                    Console.WriteLine("CreateIndex failed with
response: " + result.ToString());
                }
            }
        }
    }
    catch (MongoException ex)
    {
        Console.WriteLine("MongoDbService InitializeVectorIndex: " +
ex.Message);
        throw;
    }
}
```

3. Select the **Ask AI Assistant** (search for a recipe by name or description, or ask a question) option in the application to run a user query.

The user query is converted to an embedding using the Open AI service and the embedding model. The embedding is then sent to Azure Cosmos DB for MongoDB and is used to perform a vector search. The `VectorSearchAsync` method in the `VCoreMongoService.cs` file performs a vector search to find vectors that are close to the supplied vector and returns a list of documents from Azure Cosmos DB for MongoDB vCore.

C#

```
public async Task<List<Recipe>> VectorSearchAsync(float[] queryVector)
{
    List<string> retDocs = new List<string>();
    string resultDocuments = string.Empty;

    try
    {
        //Search Azure Cosmos DB for MongoDB vCore collection for
        similar embeddings
        //Project the fields that are needed
        BsonDocument[] pipeline = new BsonDocument[]
        {
            BsonDocument.Parse(
                @$"{{$search: {{"
                    "cosmosSearch:
                    {{ vector: [{string.Join(',',"
                    queryVector)}],
                    path: 'embedding',
                    k: {_maxVectorSearchResults}}},
                    returnStoredSource:true
                }}}
            }"""),
            BsonDocument.Parse($"{{$project: {{embedding: 0}}}}"),
        };

        var bsonDocuments = await _recipeCollection
            .Aggregate<BsonDocument>(pipeline).ToListAsync();

        var recipes = bsonDocuments
            .ToList()
            .ConvertAll(bsonDocument =>
                BsonSerializer.Deserialize<Recipe>(bsonDocument));
        return recipes;
    }
    catch (MongoException ex)
    {
        Console.WriteLine($"Exception: VectorSearchAsync():
{ex.Message}");
        throw;
    }
}
```

The `GetChatCompletionAsync` method generates an improved chat completion response based on the user prompt and the related vector search results.

C#

```
public async Task<string response, int promptTokens, int
responseTokens> GetChatCompletionAsync(string userPrompt, string
documents)
{
    try
    {
        ChatMessage systemMessage = new ChatMessage(
            ChatRole.System, _systemPromptRecipeAssistant + documents);
        ChatMessage userMessage = new ChatMessage(
            ChatRole.User, userPrompt);

        ChatCompletionsOptions options = new()
        {
            Messages =
            {
                systemMessage,
                userMessage
            },
            MaxTokens = openAIMaxTokens,
            Temperature = 0.5f, //0.3f,
            NucleusSamplingFactor = 0.95f,
            FrequencyPenalty = 0,
            PresencePenalty = 0
        };

        Azure.Response<ChatCompletions> completionsResponse =
            await
openAIClient.GetChatCompletionsAsync(openAICompletionDeployment,
options);
        ChatCompletions completions = completionsResponse.Value;

        return (
            response: completions.Choices[0].Message.Content,
            promptTokens: completions.Usage.PromptTokens,
            responseTokens: completions.Usage.CompletionTokens
        );
    }
    catch (Exception ex)
    {
        string message = $"OpenAIService.GetChatCompletionAsync():
{ex.Message}";
        Console.WriteLine(message);
        throw;
    }
}
```

The app also uses prompt engineering to ensure Open AI service limits and formats the response for supplied recipes.

```
C#  
  
//System prompts to send with user prompts to instruct the model for  
chat session  
private readonly string _systemPromptRecipeAssistant = @"  
    You are an intelligent assistant for Contoso Recipes.  
    You are designed to provide helpful answers to user questions about  
    recipes, cooking instructions provided in JSON format below.  
  
    Instructions:  
    - Only answer questions related to the recipe provided below.  
    - Don't reference any recipe not provided below.  
    - If you're unsure of an answer, say ""I don't know"" and recommend  
    users search themselves.  
    - Your response should be complete.  
    - List the Name of the Recipe at the start of your response  
    followed by step by step cooking instructions.  
    - Assume the user is not an expert in cooking.  
    - Format the content so that it can be printed to the Command Line  
    console.  
    - In case there is more than one recipe you find, let the user pick  
    the most appropriate recipe.";
```

Dimensionar o OpenAI do Azure para chat do .NET usando RAG com Aplicativos de Contêiner do Azure

Artigo • 21/05/2024

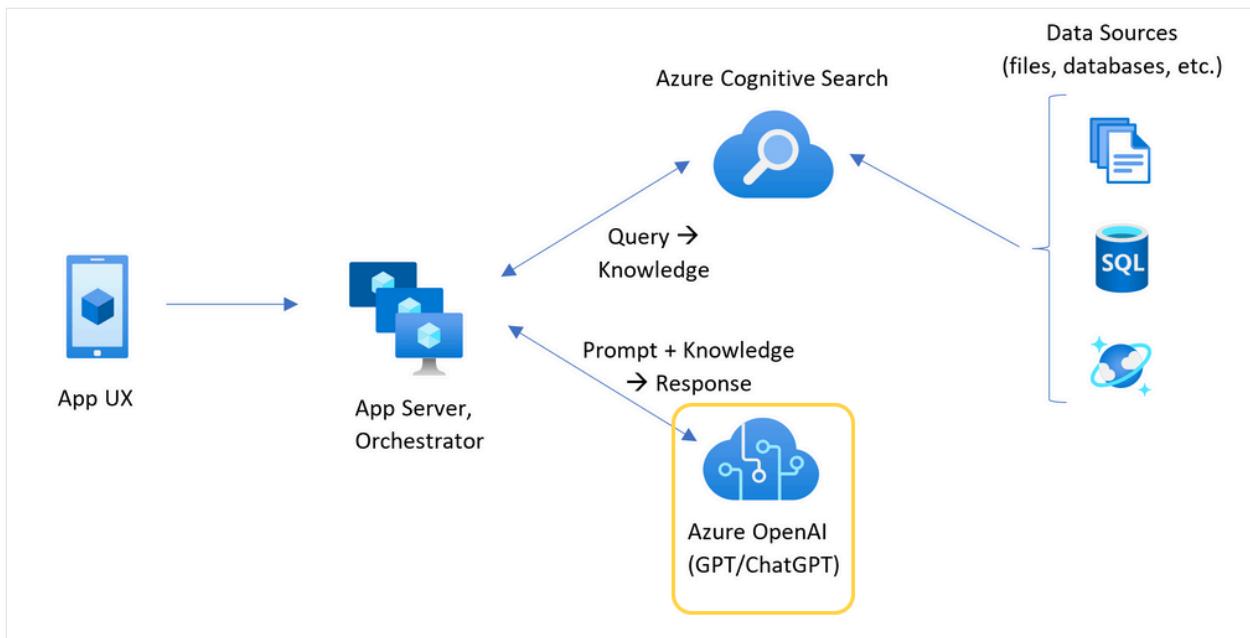
Saiba como adicionar balanceamento de carga ao seu aplicativo para estender o aplicativo de chat além dos limites de cota de modelo e token do OpenAI do Azure. Esta abordagem usa os Aplicativos de Contêiner do Azure para criar três pontos de extremidade do OpenAI do Azure, bem como um contêiner primário para direcionar o tráfego de entrada para um dos três pontos de extremidade.

Este artigo exige que você implante dois exemplos separados:

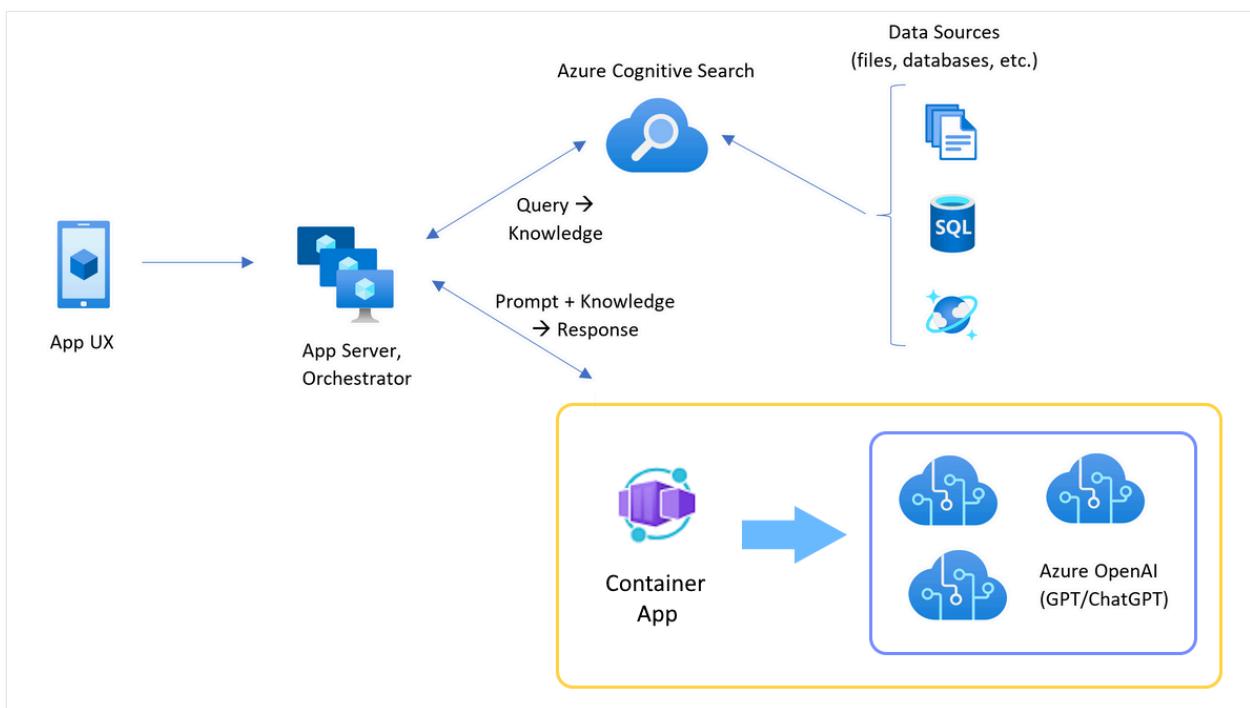
- Aplicativo de chat
 - Se você ainda não implantou o aplicativo de chat, aguarde até que o exemplo do平衡ador de carga seja implantado.
 - Se você já tiver implantado o aplicativo de chat uma vez, você alterará a variável de ambiente para dar suporte a um ponto de extremidade personalizado para o平衡ador de carga e reimplantá-lo novamente.
 - Aplicativo de chat disponível nestes idiomas:
 - [.NET](#)
 - [JavaScript](#)
 - [Python](#)
- Aplicativo balanceador de carga

Arquitetura para balanceamento de carga do OpenAI do Azure com Aplicativos de Contêiner do Azure

Como o recurso OpenAI do Azure tem limites específicos de cota de token e modelo, um aplicativo de chat usando um único recurso do OpenAI do Azure está propenso a ter falhas de conversa devido a esses limites.

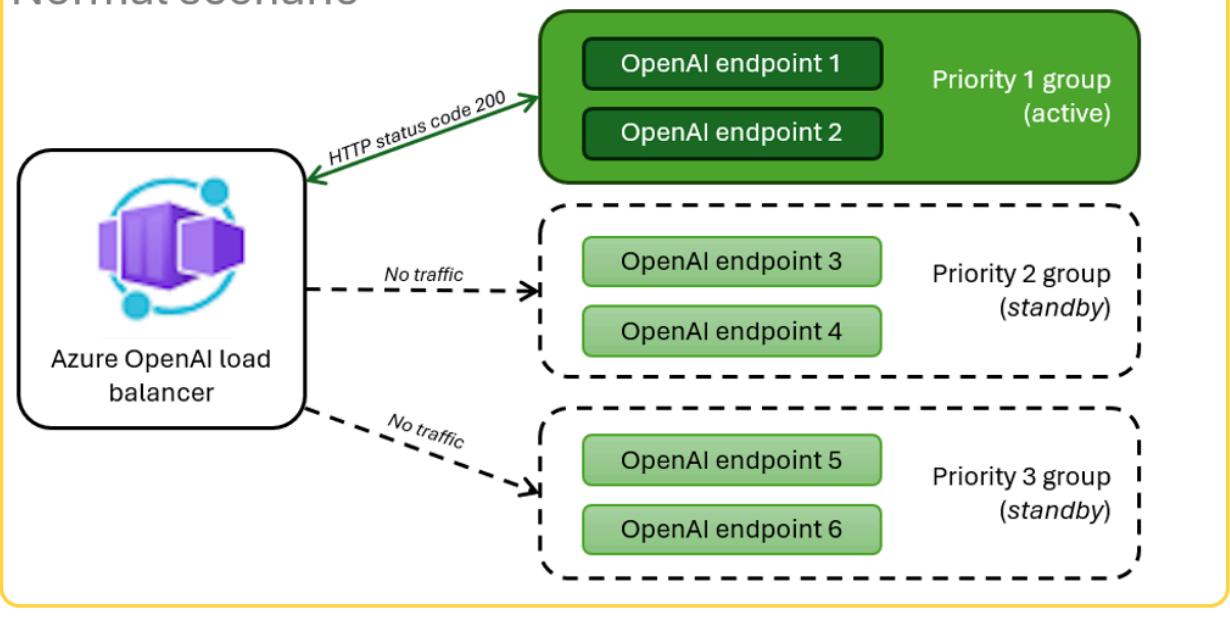


Para usar o aplicativo de chat sem atingir esses limites, use uma solução com balanceamento de carga com os Aplicativos de Contêiner do Azure. Esta solução expõe perfeitamente um único ponto de extremidade dos Aplicativos de Contêiner do Azure para o servidor de aplicativo de chat.



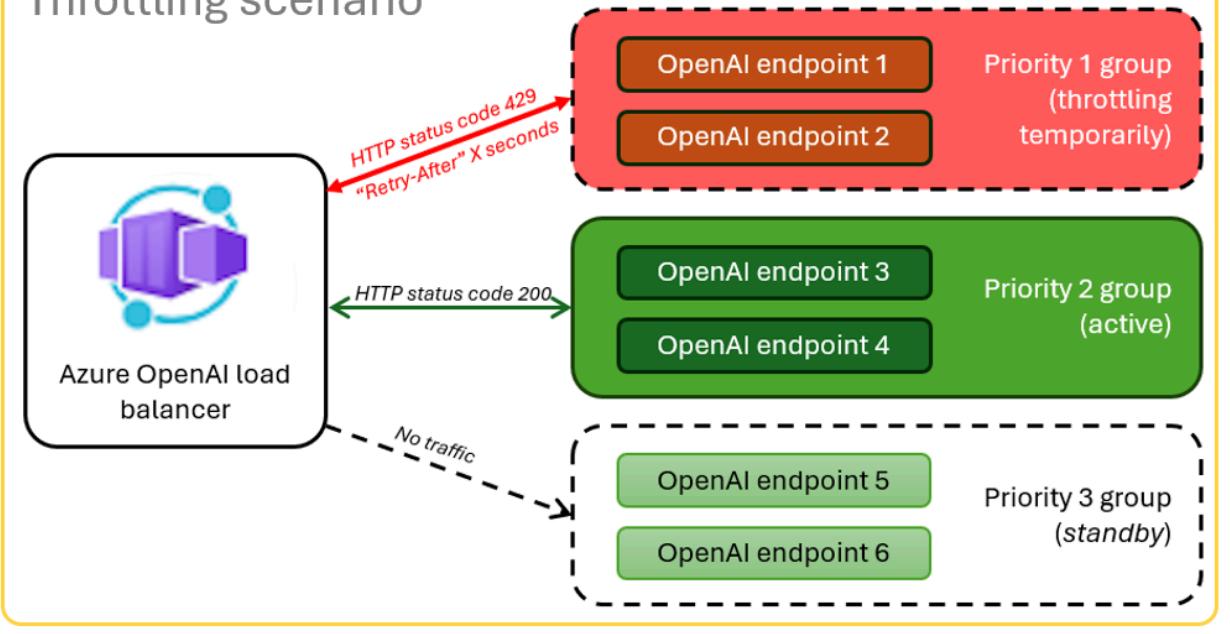
O Aplicativo de Contêiner do Azure fica na frente de um conjunto de recursos do OpenAI do Azure. O Aplicativo de Contêiner resolve dois cenários: normal e limitado. Durante um **cenário normal** em que a cota de token e modelo estão disponíveis, o recurso do OpenAI do Azure retorna um 200 de volta por meio do Aplicativo de Contêiner e do Servidor de Aplicativos.

Normal scenario



Quando um recurso está em um **cenário limitado**, como devido a limites de cota, o aplicativo de contêiner do Azure pode tentar novamente um recurso do OpenAI do Azure diferente imediatamente para realizar a solicitação de aplicativo de chat original.

Throttling scenario



Pré-requisitos

- Assinatura do Azure. [Crie um gratuitamente](#) ↗
- Acesso permitido ao OpenAI do Azure na assinatura do Azure desejada.

No momento, o acesso a esse serviço é permitido somente por aplicativo. Você deve [solicitar acesso](#) ao OpenAI do Azure.

- Os [contêineres de desenvolvimento](#) estão disponíveis para ambos os exemplos, com todas as dependências necessárias para concluir este artigo. Você pode executar os contêineres de desenvolvimento em Codespaces do GitHub (em um navegador) ou localmente usando o Visual Studio Code.

Codespaces (recomendado)

- Somente é necessário ter uma [conta do GitHub](#) para usar o CodeSpaces

Abrir o aplicativo de exemplo de平衡ador local de aplicativos de contêiner

Codespaces (recomendado)

O [GitHub Codespaces](#) executa um contêiner de desenvolvimento gerenciado pelo GitHub com o [Visual Studio Code para Web](#) como interface do usuário. Para o ambiente de desenvolvimento mais simples, use os Codespaces do GitHub para que você tenha as ferramentas e dependências de desenvolvedor corretas pré-instaladas para concluir este artigo.



[Open in GitHub Codespaces](#)

ⓘ Importante

Todas as contas do GitHub podem usar Codespaces por até 60 horas gratuitas por mês com 2 instâncias principais. Para saber mais, confira [Armazenamento e horas por núcleo incluídos mensalmente no GitHub Codespaces](#).

Implantar o balanceador de carga dos Aplicativos de Contêiner do Azure

1. Entre na CLI do Desenvolvedor do Azure para fornecer autenticação às etapas de provisionamento e implantação.

```
Bash
```

```
azd auth login --use-device-code
```

2. Defina uma variável de ambiente para usar a autenticação da CLI do Azure para a etapa de pós-provisionamento.

```
Bash
```

```
azd config set auth.useAzCliAuth "true"
```

3. Implante o aplicativo do balanceador de carga.

```
Bash
```

```
azd up
```

Você precisará selecionar uma assinatura e uma região para a implantação. Estas não precisam ser a mesma assinatura e região que o aplicativo de chat.

4. Aguarde a implantação ser concluída antes de continuar.

Obter o ponto de extremidade de implantação

1. Use o seguinte comando para exibir o ponto de extremidade implantado para o aplicativo Contêiner do Azure.

```
Bash
```

```
azd env get-values
```

2. Copie o valor `CONTAINER_APP_URL`. Você o usará na próxima seção.

Reimplantar o aplicativo de chat com o ponto de extremidade do平衡ador de carga

Elas são concluídas no exemplo do aplicativo de chat.

Implantação inicial

1. Abra o contêiner de desenvolvimento do aplicativo de chat usando uma das opções a seguir.

[+] Expandir a tabela

Idioma	Codespaces	Visual Studio Code
.NET	 Open in GitHub Codespaces 	 Open 
JavaScript	 Open in GitHub Codespaces 	 Open 
Python	 Open in GitHub Codespaces 	 Open 

2. Entre na Azure Developer CLI (AZD).

```
Bash
azd auth login
```

Conclua as instruções de entrada.

3. Crie um ambiente do AZD com um nome como `chat-app`.

```
Bash
azd env new <name>
```

4. Adicione a variável de ambiente a seguir, que informa ao back-end do aplicativo Chat para usar uma URL personalizada para as solicitações do OpenAI.

```
Bash
azd env set OPENAI_HOST azure_custom
```

5. Adicione a variável de ambiente a seguir, substituindo `<CONTAINER_APP_URL>` pela URL da seção anterior. Esta ação informa ao back-end do aplicativo de chat qual é o valor da URL personalizada para a solicitação do OpenAI.

```
Bash
```

```
azd env set AZURE_OPENAI_CUSTOM_URL <CONTAINER_APP_URL>
```

6. Implante o aplicativo de chat.

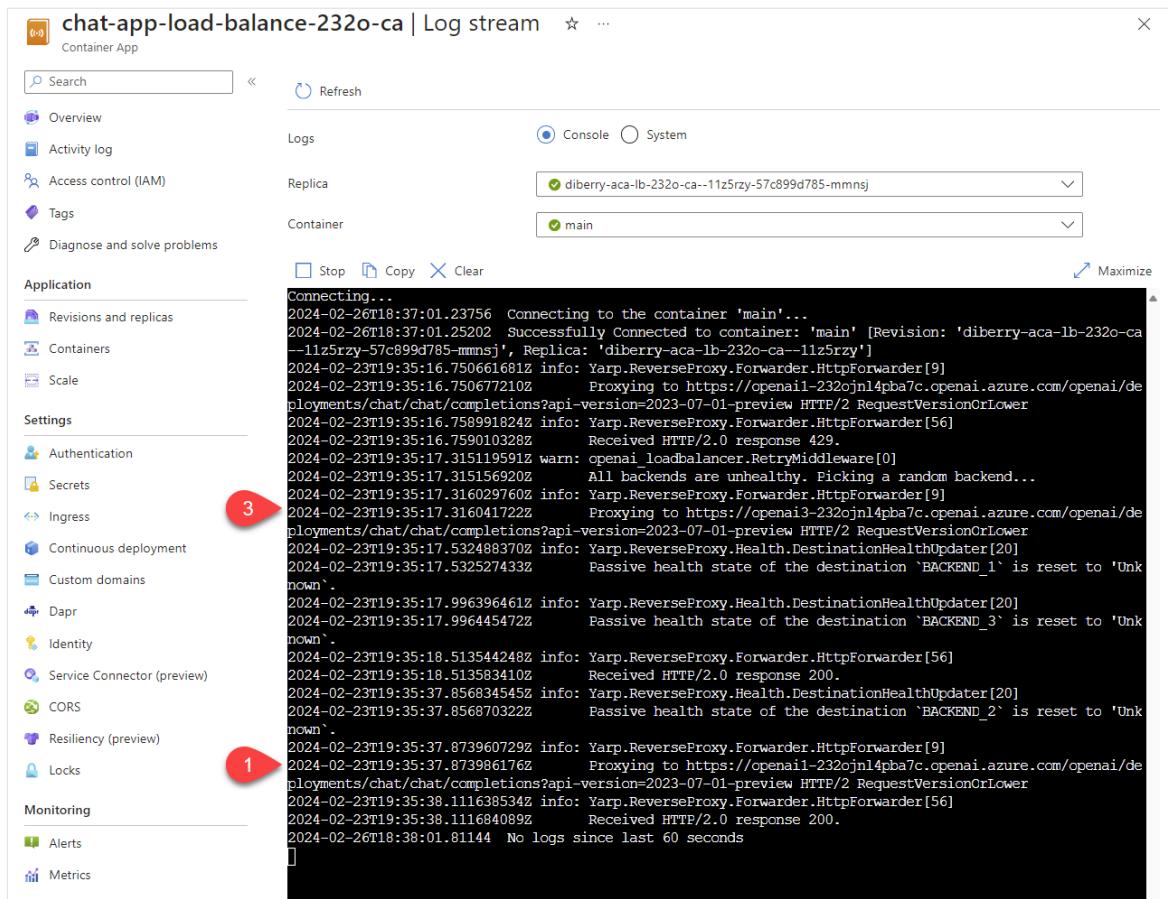
Bash

```
azd up
```

Agora você pode usar o aplicativo de chat com a confiança de que ele foi criado para ser escalado entre muitos usuários sem ficar sem cota.

Transmitir logs para ver os resultados do balanceador de carga

1. No [portal do Azure](#), pesquise seu grupo de recursos.
2. Na lista de recursos no grupo, selecione o recurso Aplicativo de Contêiner.
3. Selecione **Monitoramento** → **fluxo de log** para exibir o log.
4. Use o aplicativo de chat para gerar tráfego no log.
5. Procure os logs, que fazem referência aos recursos do OpenAI do Azure. Cada um dos três recursos tem sua identidade numérica no comentário de log começando com `Proxying to https://openai3`, em que `3` indica o terceiro recurso do OpenAI do Azure.



6. Conforme você usa o aplicativo de chat, quando o平衡ador de carga recebe o status de que a solicitação excedeu a cota, o平衡ador de carga gira automaticamente para outro recurso.

Configurar a cota de tokens por minuto (TPM)

Por padrão, cada uma das instâncias do OpenAI no平衡ador de carga será implantada com capacidade de 30.000 TPM (tokens por minuto). Você pode usar o aplicativo de chat tendo a confiança de que ele foi criado para ser dimensionado entre muitos usuários sem ficar sem cota. Altere esse valor quando:

- Se ocorrerem erros de capacidade de implantação: reduza esse valor.
- Ao planejar uma capacidade mais alta, aumente o valor.

1. Use o comando a seguir para alterar o valor.

```
Bash
azd env set OPENAI_CAPACITY 50
```

2. Reimplante o平衡ador de carga.

```
Bash
```

```
azd up
```

Limpar os recursos

Quando terminar de usar o aplicativo de chat e o balanceador de carga, limpe os recursos. Os recursos do Azure criados neste artigo são cobrados para sua assinatura do Azure. Se você não espera precisar desses recursos no futuro, exclua-os para evitar incorrer em mais encargos.

Limpar recursos do aplicativo de chat

Retorne ao artigo do aplicativo de chat para limpar esses recursos.

- [.NET](#)
- [JavaScript](#)
- [Python](#)

Limpar recursos do balanceador de upload

Execute o seguinte comando do Azure Developer CLI para excluir os recursos do Azure e remover o código-fonte:

```
Bash
```

```
azd down --purge --force
```

As opções fornecem:

- `purge`: os recursos excluídos são limpos imediatamente. Isso permite reutilizar o TPM do OpenAI do Azure.
- `force`: a exclusão ocorre silenciosamente, sem exigir o consentimento do usuário.

Limpar GitHub Codespaces

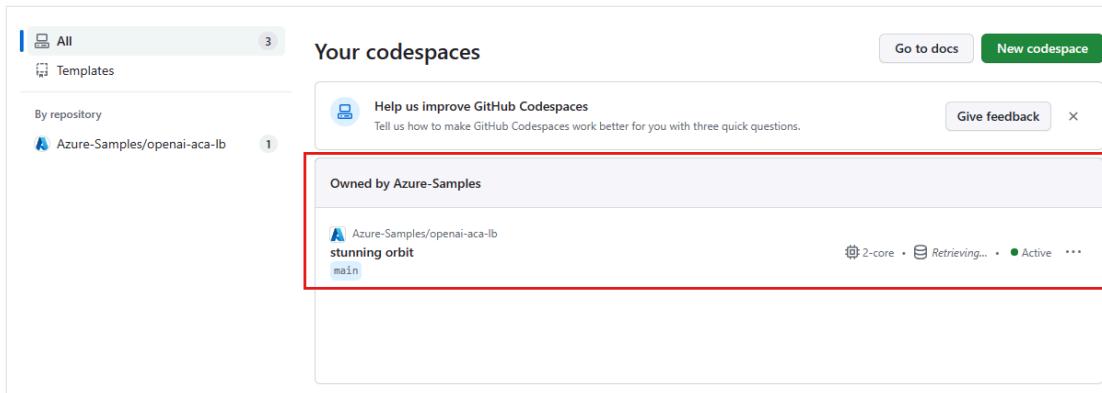
Codespaces do GitHub

A exclusão do ambiente GitHub Codespaces garante que você possa maximizar a quantidade de horas gratuitas por núcleo que você tem direito na sua conta.

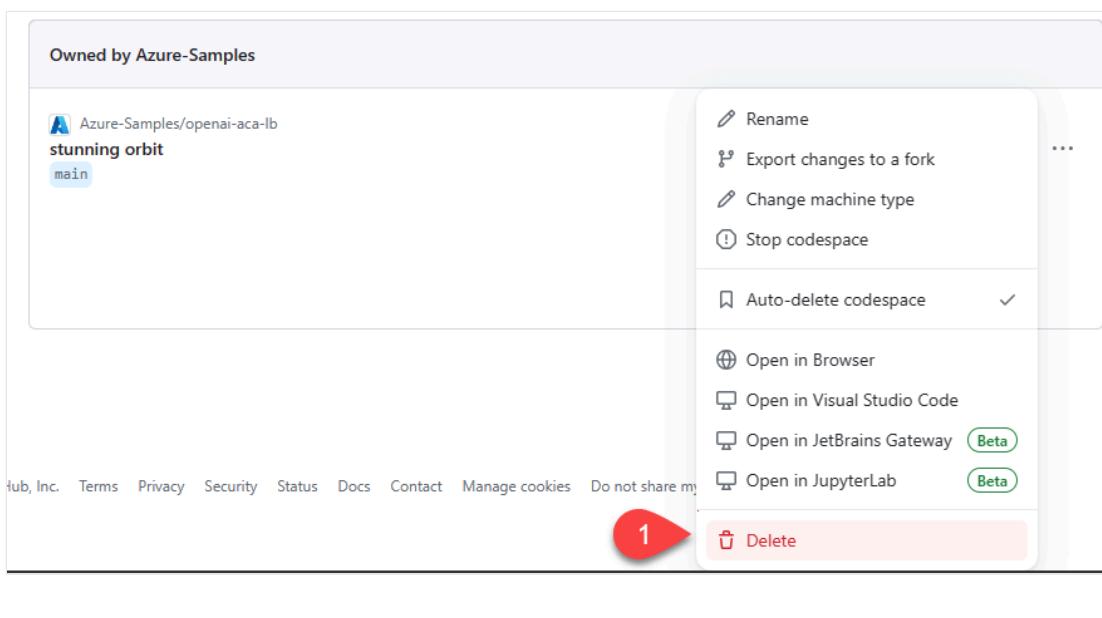
ⓘ Importante

Para saber mais sobre os direitos da sua conta do GitHub, confira [O GitHub Codespaces inclui mensalmente armazenamento e horas de núcleo ↗](#).

1. Entre no painel do GitHub Codespaces (<https://github.com/codespaces> ↗).
2. Localize seus Codespaces atualmente em execução, originados do repositório [azure-samples/openai-aca-lb](#) ↗ do GitHub.



3. Abra o menu de contexto do codespace e selecione Excluir.



Obter ajuda

Se você tiver problemas para implantar o balanceador de carga do Gerenciamento de API do Azure, registre o seu problema nos [Problemas](#) ↗ do repositório.

Código de exemplo

Os exemplos usados neste artigo incluem:

- [Aplicativo de chat do .NET com RAG ↗](#)
- [Balanceador de carga com Aplicativos de Contêiner do Azure ↗](#)

Próxima etapa

- Usar o [Teste de Carga do Azure](#) para executar teste de carga do seu aplicativo de chat

Colaborar conosco no GitHub

A fonte deste conteúdo pode ser encontrada no GitHub, onde você também pode criar e revisar problemas e solicitações de pull. Para obter mais informações, confira o [nossa guia para colaboradores](#).

.NET

Comentários do .NET

O .NET é um projeto código aberto. Selecione um link para fornecer comentários:

 [Abrir um problema de documentação](#)

 [Fornecer comentários sobre o produto](#)