

Enabling Real-Time Intelligence with Custom Skills on AI chatbots

Agenda

- Introduction (5 mins)
- LLM Basics (5 mins)
- AI Skills Intro (5 mins)
- Live Demo (5 mins)
- Code Walkthrough (20 mins)
- CoPilot Studio (5 mins)
- Q&A

Speakers

Balaji Iyer



Fawad Shaikh



Microsoft Atlanta



Teams

- Azure
- M365
- Finance
- LinkedIn

Community Impact

- ERG
- ATL residents' exposure to tech
- University Relations / Internships



LLM

Basics

6 things we know LLM is really good at



Generation



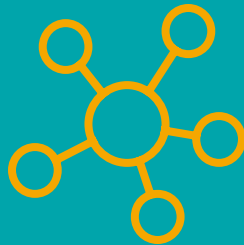
Transformation



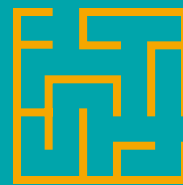
Conversation



Summarization



Classification



Completion

LLMs predict what words should exist in a sequence, they don't think through things or understand language

What LLMs **DO**

- **Personalize** (everything becomes a market of one)
- **Proactive** (does actions for you)
- **Predict** (completes sentences based on context)

What LLMs **DO NOT DO**

- **Understand** (they do not actually know about the text they are generating)
- **Reason** (Getting better with new models)
- **Judge** (they do not have morals or ethics beyond the guardrails people add)

LLM = extremely advanced
auto-complete

AI Agents & Skills Intro

What is an AI Agent?

- An AI Agent is a software application that uses AI / LLM for a specific purpose.
- "Single query" models
 - Ex: Siri, Alexa, Google Home
 - Take in speech, convert your voice into text, feed that text into their respective AI systems.
- "Cache it and query later" models
 - Ex: Google Photos
 - Takes in data in batches (photos), processes data by adding metadata (location, date, etc.) and classifying data (people, objects, animals, etc.), then allows for intelligent querying later (photos taken here, at specific time, with specific people)
- "Chatbot" models
 - Ex: ChatGPT
 - Responds to a provided conversational context

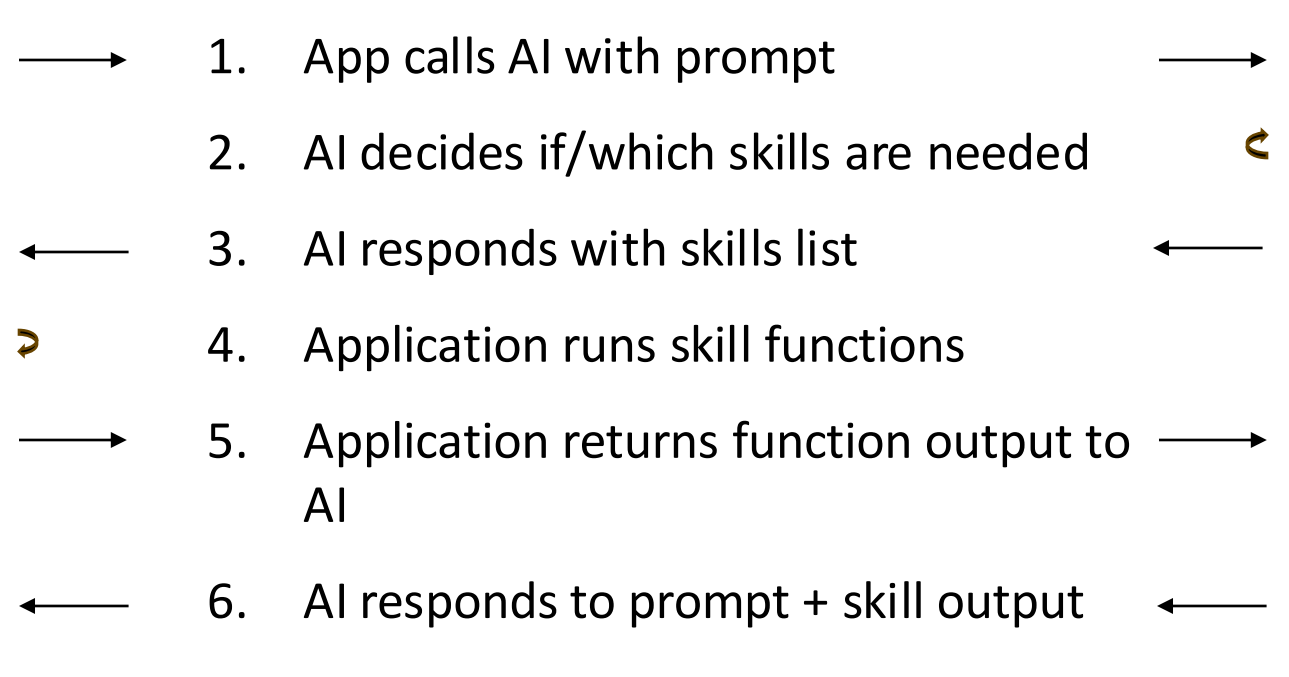
What is a Skill?

- The ability of an AI agent to perform tasks by invoking pre-defined functions / API calls.
- Skills have many different names:
 - Skills, Tools, Functions, Plugins, Add-ins, Extensions
 - They all mean the same thing.
- Models that support Plugins: [Berkeley Function Calling Leaderboard](#)
 - OpenAI
 - Mistral
 - Claude
 - Llama

Skill Flow

Application

AI



Pre-Requisites

Decide on an AI Model

- Azure AI Studio (ai.azure.com)
 - Allows easy to use/manage AI deployments
 - Thousands of different models. Including:
 - OpenAI
 - Llama
 - Mistral
 - Designed for Enterprise Privacy
 - Your inputs/outputs are never used to train any model.
 - Your inputs/outputs are never shared with any 3rd party.

| | | |
|--------------------------|--|----|
| <input type="checkbox"/> |  Azure OpenAI | 19 |
| <input type="checkbox"/> |  Microsoft | 16 |
| <input type="checkbox"/> |  Meta | 29 |
| <input type="checkbox"/> |  Mistral | 12 |

Use Any Language You Want

- Myth: You have to use Python to enable AI in your software
 - Almost all AI APIs use REST.
 - Any programming language that can connect to REST API can be used to add LLM AIs in your software application.
 - Our demo app uses C# Aspire Framework for the backend and React (Typescript) for the frontend.
- Aspire Framework
 - Opinionated framework to build modern microservice based applications on .Net.
 - [.NET Aspire overview - .NET Aspire | Microsoft Learn](#)

Demo

Code Walkthrough

aka.ms/atldevconskill

Explore the UI

- Inputs:
 - City
 - State
- Outputs[]:
 - Timeframe
 - TemperatureF
 - Summary
 - TemperatureC

Weather Forecast

Check the weather for your city

City

State

Alpharetta

GA

Get Weather

| Timeframe | Temp. (C) | Temp. (F) | Summary |
|----------------|-----------|-----------|--------------|
| Tonight | 17 | 64 | Mostly Clear |
| Thursday | 30 | 86 | Mostly Sunny |
| Thursday Night | 18 | 65 | Mostly Clear |
| Friday | 31 | 89 | Sunny |
| Friday Night | 17 | 64 | Mostly Clear |
| Saturday | 32 | 90 | Sunny |

Explore the API: /weatherforecast

- Inputs:
 - City
 - State
- Outputs[]:
 - Order
 - Name
 - TemperatureF
 - Summary
 - Details
 - TemperatureC

The screenshot shows the Swagger UI for the `/weatherforecast` endpoint. The interface includes a browser window with the URL `https://localhost:7437/swagger/index.html`. The left sidebar shows the Swagger UI tab. The main area displays the `GET /weatherforecast` endpoint with two required query parameters: `city` (string) and `state` (string). The `city` parameter is set to `Alpharetta` and the `state` parameter is set to `GA`. Below the parameters, there is an `Execute` button and a `Clear` button. The `Responses` section shows the `200` response with a JSON body. The JSON body contains an array of weather forecast objects, each with `order`, `name`, `temperatureF`, `summary`, `details`, and `temperatureC` properties. The response headers are also displayed, including `content-type: application/json; charset=utf-8`, `date: Wed, 18 Sep 2024 20:11:07 GMT`, and `server: Kestrel`. At the bottom, there is a table with columns `Code`, `Description`, and `Links`, showing the `200 OK` response with a media type of `application/json`.

```
curl -X 'GET' \
  'https://localhost:7437/weatherforecast?city=Alpharetta&state=GA' \
  -H 'accept: application/json'
```

```
{
  "order": 1,
  "name": "This Afternoon",
  "temperatureF": 82,
  "summary": "Partly Sunny",
  "details": "Partly sunny, with a high near 82. Northwest wind around 5 mph.",
  "temperatureC": 27
},
{
  "order": 2,
  "name": "Tonight",
  "temperatureF": 64,
  "summary": "Partly Cloudy",
  "details": "Partly cloudy, with a low around 64. Northeast wind 0 to 5 mph.",
  "temperatureC": 17
},
{
  "order": 3,
  "name": "Thursday",
  "temperatureF": 86,
  "summary": "Mostly Sunny",
  "details": "Mostly sunny, with a high near 86. Northeast wind 0 to 5 mph.",
  "temperatureC": 30
}
```

```
content-type: application/json; charset=utf-8
date: Wed, 18 Sep 2024 20:11:07 GMT
server: Kestrel
```

| Code | Description | Links |
|------|-------------|----------|
| 200 | OK | No links |

Explore the Code: GetForecastAsync

- Inputs:
 - city
 - state
- Outputs:
 - ForecastResponse[]

```
4 references
public async Task<IEnumerable<WeatherService.ForecastResponse>> GetForecastsAsync(string city, string state)
{
    var geoLocation = await _locationService.GetGeoLocationAsync(city, state);

    if (geoLocation == null)
    {
        return Enumerable.Empty<WeatherService.ForecastResponse>();
    }

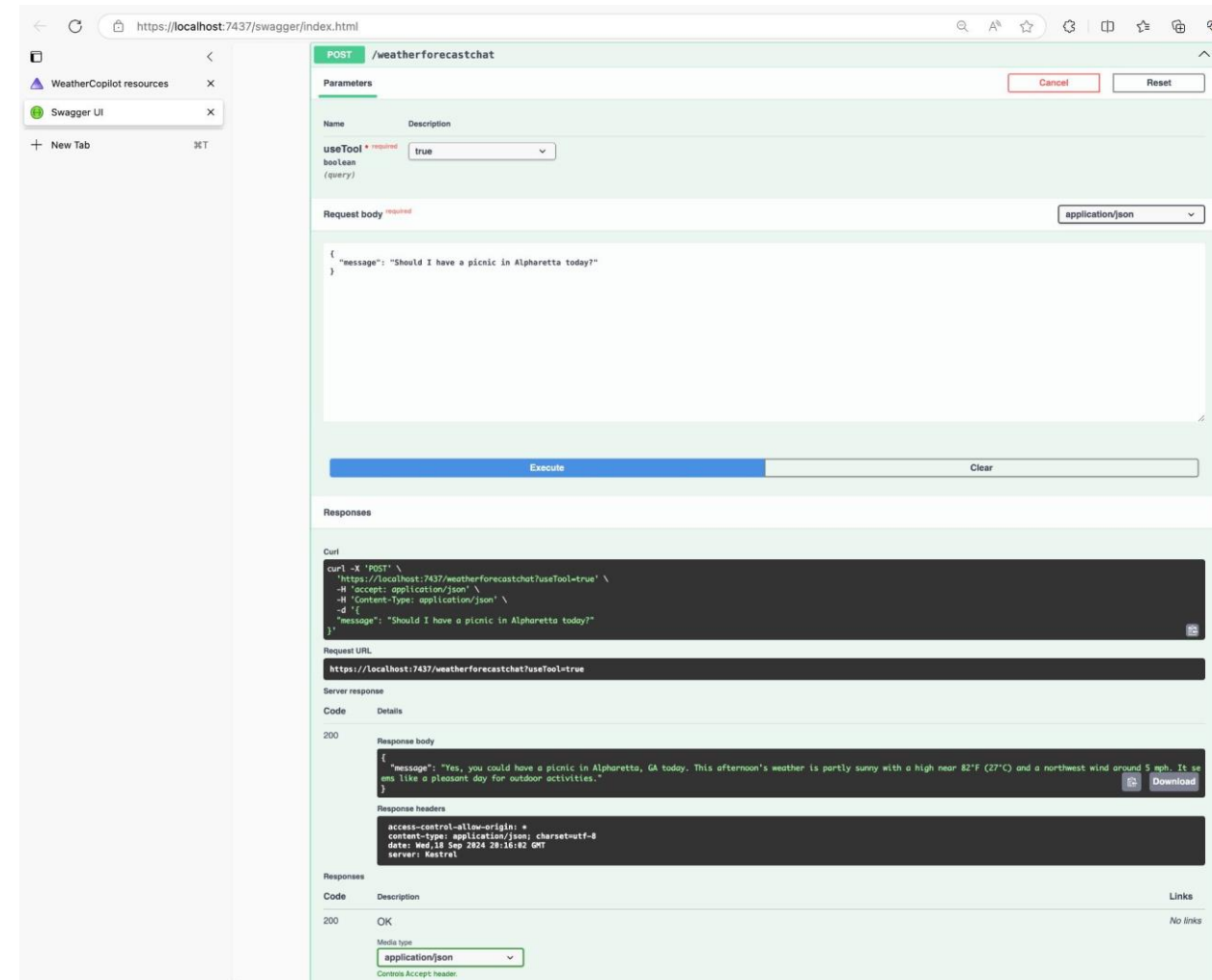
    var latitude = double.Parse(geoLocation?.Latitude ?? "0.0");
    var longitude = double.Parse(geoLocation?.Longitude ?? "0.0");

    return await _weatherService.CallWeatherServiceAsync(latitude, longitude);
}

public record ForecastResponse(int Order, string Name, int TemperatureF, string Summary, string Details)
{
    0 references
    public int TemperatureC => (int)((TemperatureF - 32) * (5.0 / 9.0));
}
```

Explore the API: /weatherforecastchat

- Inputs:
 - message
- Outputs:
 - message



Explore the Code: CompleteChatAsync

- Inputs:
 - prompt
 - useTool
- Outputs:
 - ChatMessage

```
public async Task<WebForecast.ChatMessage> CompleteChatAsync(string prompt, bool useTool = false)
{
    List<ChatMessage> conversationMessages = new List<ChatMessage> {
        // System messages represent instructions or other guidance about how the assistant should behave
        new SystemChatMessage(@"You are an assistant that helps people answer questions using details of the weather in their location. (City and State).
        You are limited to American cities only. Keep your responses clear and concise."),
        // User messages represent user input, whether historical or the most recent input
        new UserChatMessage(prompt)
    };
}
```

```
public record ChatMessage(string Message);
```

Explore the Code: CompleteChatAsync

- Define the tool

```
ChatTool getWeatherForecastTool = ChatTool.CreateFunctionTool(  
    functionName: nameof(_forecastService.GetForecastsAsync),  
    functionDescription: "Get the current and upcoming weather forecast for a given city and state",  
    functionParameters: BinaryData.FromString("""  
        {  
            "type": "object",  
            "properties": {  
                "city": {  
                    "type": "string",  
                    "description": "The city to get the weather for. eg: Miami or New York"  
                },  
                "state": {  
                    "type": "string",  
                    "description": "The state the city is in. eg: FL or Florida or NY or New York"  
                }  
            },  
            "required": [ "city", "state" ]  
        }  
        """)  
);
```

Explore the Code: CompleteChatAsync

- Call LLM with Tool

```
ChatCompletionOptions options = new()
{
    Tools = { },
};

if (useTool)
{
    options.Tools.Add(getWeatherForecastTool);
}

ChatCompletion completion = _chatClient.CompleteChat(conversationMessages, options);
```


Explore the Code: CompleteChatAsync

- Check if LLM decided to use a tool

```
if (completion.ToolCalls.Count > 0)
{
    // Important to add the completion to the conversation messages so next completion can be aware of the previous completion
    conversationMessages.Add(new AssistantChatMessage(completion));
    foreach (var toolCall in completion.ToolCalls)
    {
        if (toolCall.FunctionName == nameof(_forecastService.GetForecastsAsync))
        {

```

Explore the Code: CompleteChatAsync

- Run Tool(s)
- Add response to messages
- Call LLM again

```
foreach (var toolCall in completion.ToolCalls)
{
    if (toolCall.FunctionName == nameof(_forecastService.GetForecastsAsync))
    {
        using JsonDocument argumentsDocument = JsonDocument.Parse(toolCall.FunctionArguments);
        if (!argumentsDocument.RootElement.TryGetProperty("city", out JsonElement cityElement)
            || !argumentsDocument.RootElement.TryGetProperty("state", out JsonElement stateElement))
        {
            return new WebForecast.ChatMessage("Please provide a city and state to get the weather forecast.");
        }

        var city = cityElement.GetString() ?? "New York";
        var state = stateElement.GetString() ?? "NY";

        var forecasts = await _forecastService.GetForecastsAsync(city, state);

        var forecastString = new StringBuilder();
        foreach (var forecast in forecasts)
        {
            forecastString.AppendLine(forecast.ToString());
        }

        // Add the concatenated forecast string as a single ToolChatMessage
        conversationMessages.Add(new ToolChatMessage(toolCall.Id, forecastString.ToString()));
    }
}

ChatCompletion finalCompletion = _chatClient.CompleteChat(conversationMessages, options);
```

Debug Walkthrough

Chat Debug Walkthrough

- Build the Messages to send to LLM

```
{
  "Messages": [
    {
      "Content": [
        {
          "Kind": "System",
          "Text": "You are an assistant that helps people answer questions using"
        }
      ]
    },
    {
      "Content": [
        {
          "Kind": "User",
          "Text": "Is it going to rain in Alpharetta this weekend?"
        }
      ]
    }
  ]
}
```

Chat Debug Walkthrough

- Create Options for Tools

```
{
  "Options": {
    "Tools": [
      {
        "FunctionName": "GetForecastsAsync",
        "FunctionDescription": "Get the current and upcoming weather forecast for a given city and state"
      }
    ]
  }
}
```

Chat Debug Walkthrough

- Assistant determines what tools (if any) need to be called

```
{
  "CreatedAt": "2024-09-19T21:31:10+00:00",
  "FinishReason": "ToolCalls",
  "ContentTokenLogProbabilities": [],
  "RefusalTokenLogProbabilities": [],
  "Role": "Assistant",
  "Content": [],
  "ToolCalls": [
    {
      "Kind": {},
      "FunctionName": "GetForecastsAsync",
      "FunctionArguments": "{\u0022city\u0022:\u0022Alpharetta\u0022,\u0022state\u0022:\u0022GA\u0022}",
      "Id": "call_kK8gJRetBa9lfc5TV7psCWT9"
    }
  ],
  "Id": "chatcmpl-A9J1yqouUEShXar5PMEZEU2lMFbcQ",
  "Model": "gpt-4",
  "SystemFingerprint": "fp_5b26d85e12",
  "Usage": {
    "OutputTokens": 23,
    "InputTokens": 147,
    "TotalTokens": 170
  }
}
```

Chat Debug Walkthrough

- Append Tool Call and Response to Conversation

```
[
  {
    "Content": [
      {
        "Kind": "System",
        "Text": "You are an assistant that helps people answer questions using details of the weather in their location. (City and State). \n"
      }
    ]
  },
  {
    "Content": [
      {
        "Kind": "User",
        "Text": "Is it going to rain in Alpharetta this weekend?"
      }
    ]
  },
  {
    "Content": [
      {
        "Kind": "Assistant",
        "ToolCalls": [
          {
            "FunctionName": "GetForecastsAsync",
            "FunctionArguments": "{\u0022city\u0022:\u0022Alpharetta\u0022,\u0022state\u0022:\u0022GA\u0022}",
            "Id": "call_kK8gJRetBa9lfc5TV7psCWT9"
          }
        ]
      }
    ]
  },
  {
    "Content": [
      {
        "Kind": "Tool",
        "Text": "ForecastResponse { Order = 1, Name = This Afternoon, TemperatureF = 85, Summary = Mostly Sunny, Details = Mostly sunny, with a hig"
      }
    ]
  }
]
```

Chat Debug Walkthrough


- Final response with LLM using the Tool data for the response

```
{
  "CreatedAt": "2024-09-19T21:31:12+00:00",
  "FinishReason": "Stop",
  "ContentTokenLogProbabilities": [],
  "RefusalTokenLogProbabilities": [],
  "Role": "Assistant",
  "Content": [
    {
      "Kind": "Assistant",
      "Text": "No, it is not going to rain in Alpharetta, GA this weekend."
    }
  ],
  "ToolCalls": [],
  "Id": "chatcmpl-A9J20iQq1MpQMzmlRk6Rh44VzgRx3",
  "Model": "gpt-4",
  "SystemFingerprint": "fp_5b26d85e12",
  "Usage": {
    "OutputTokens": 48,
    "InputTokens": 861,
    "TotalTokens": 909
  }
}
```


CoPilot Studio

CoPilot Studio

Language: **English (en-US)**

 [Edit language](#)

Name

Give your custom copilot a descriptive name so it's easy to identify. You can change this later if you need to.

Weather Copilot



 [Change icon](#)

Used to represent the copilot. Icon should be in PNG format and less than 30 KB in size.

Description

Use your own words to describe what your copilot should help with, including your audience and end goal.

Your go-to assistant for getting weather forecast

Instructions

Direct the behavior of the copilot, including its tasks and how it completes them.

You should remain friendly and polite at all times. Do not answer questions that are not related to the weather.

Connect your data

Set up the data source connections that will enable your copilot to complete tasks and access information



MSN Weather

[Set up connection](#)  

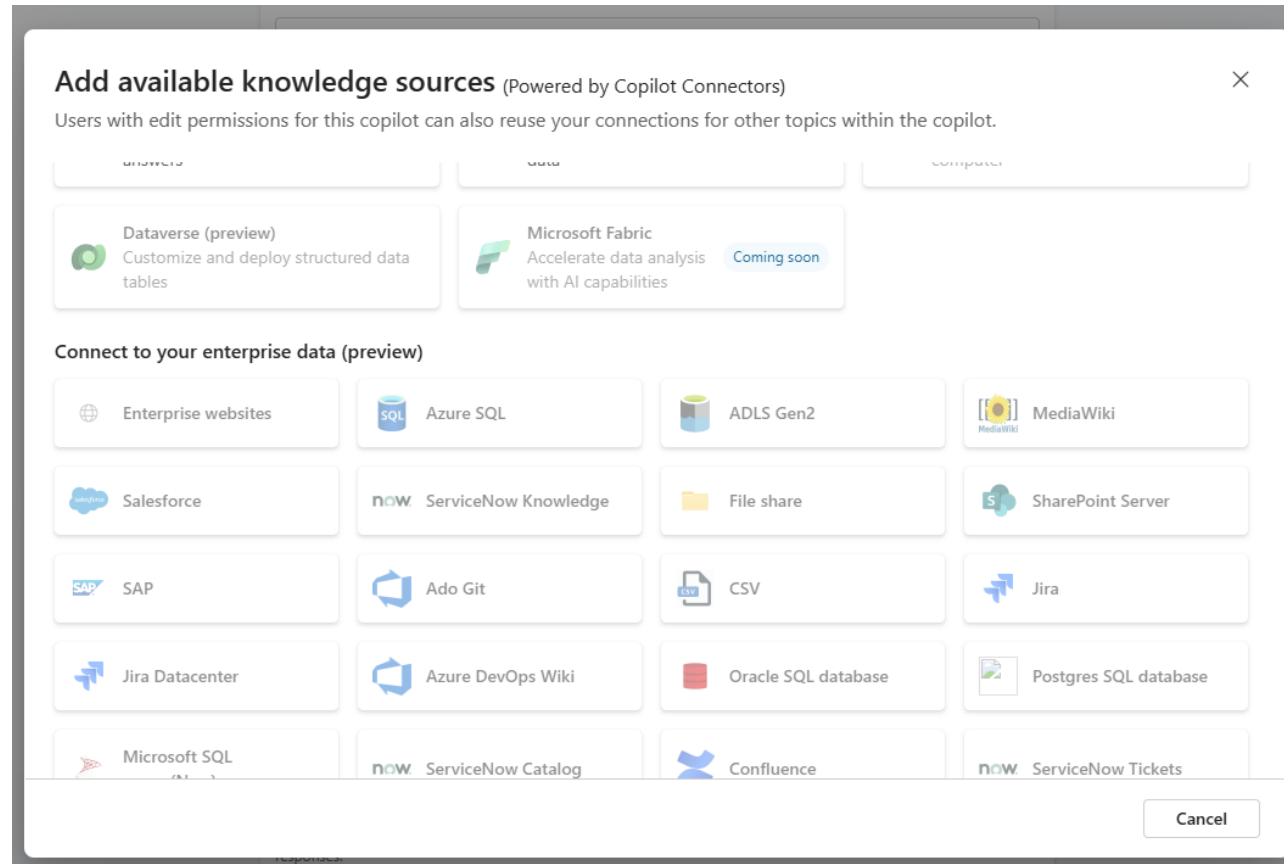
Knowledge

[+ Add knowledge](#)

Add data, files, and other resources that your copilot will use to learn. These sources form the basis for your copilot's responses.

Review [supplemental terms](#) to learn more about the templates in preview. You are responsible for complying with the terms applicable to the public URLs listed above in Knowledge. See the URLs for terms.

Custom Data Connectors



aka.ms/atlddevconskill

Thank
you