Microsoft Semantic Kernel

The Easy Way To Add LLM AI To Your App

Local LLM Development with Semantic Kernel

DALL-E prompt: small team giving a ted talk to a large audience in a theaterfast

aka.ms/semantic-kernel

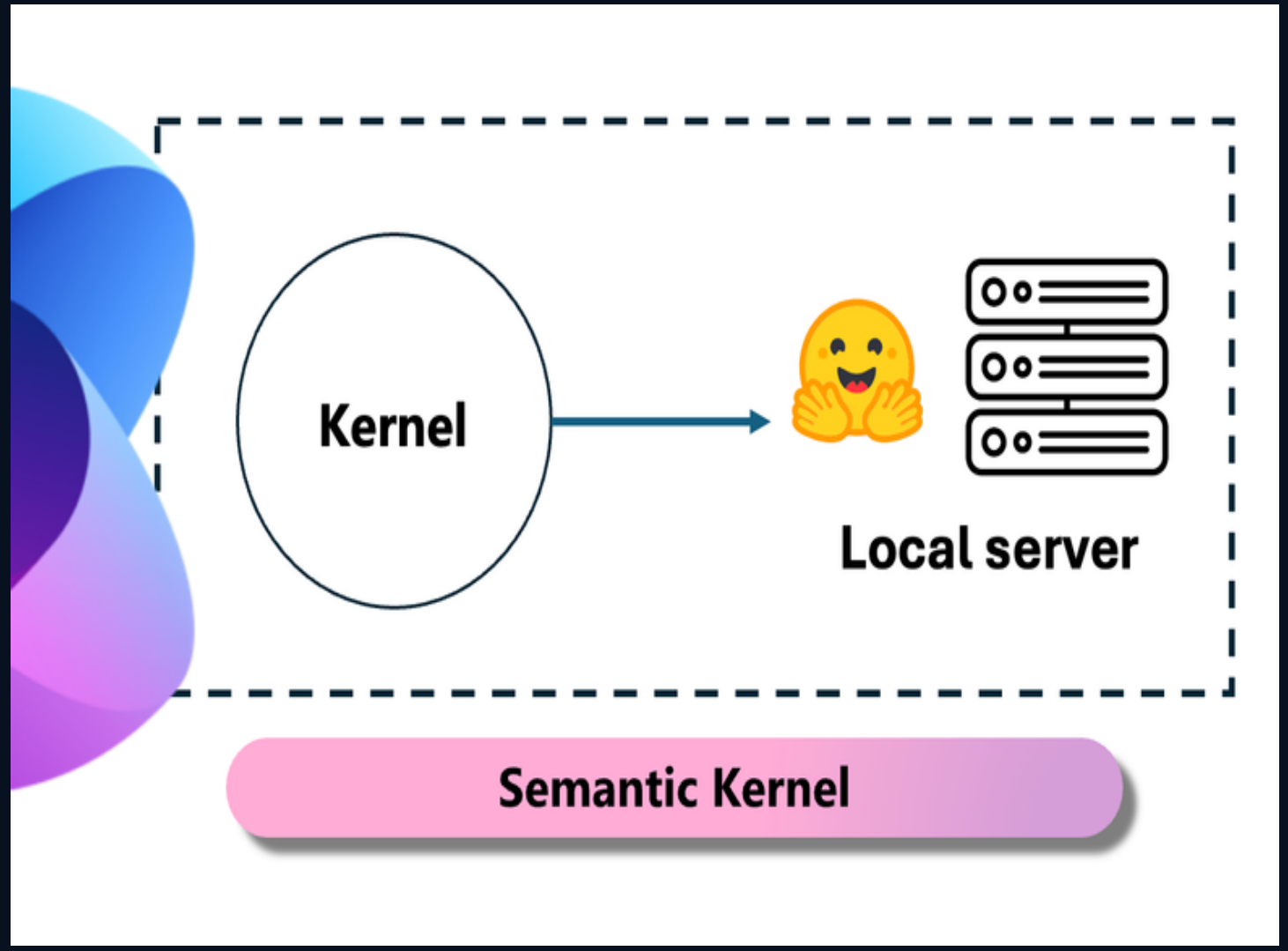
Agenda

1. Deploy a lightweight LLM
2. Sematic Kernel
3. Test some basic functionality

What you will learn

Local LLM Development

You will download your own LLM, and create a simple AI Application on your PC



1

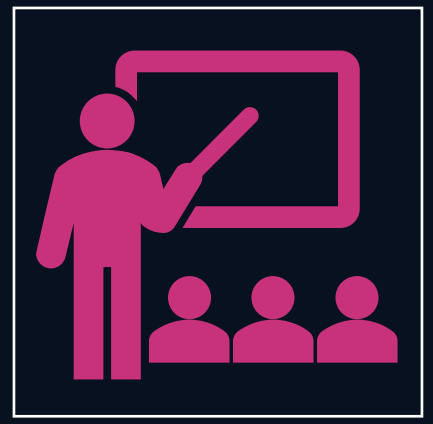
Deploy your own LLM

- We deploy a LLM on your PC
- We will meet HugginFace
- We will introduce WASM

LLM

Large Language Models

A **large language model** (**LLM**) is a computational model capable of language generation or other natural language processing tasks.



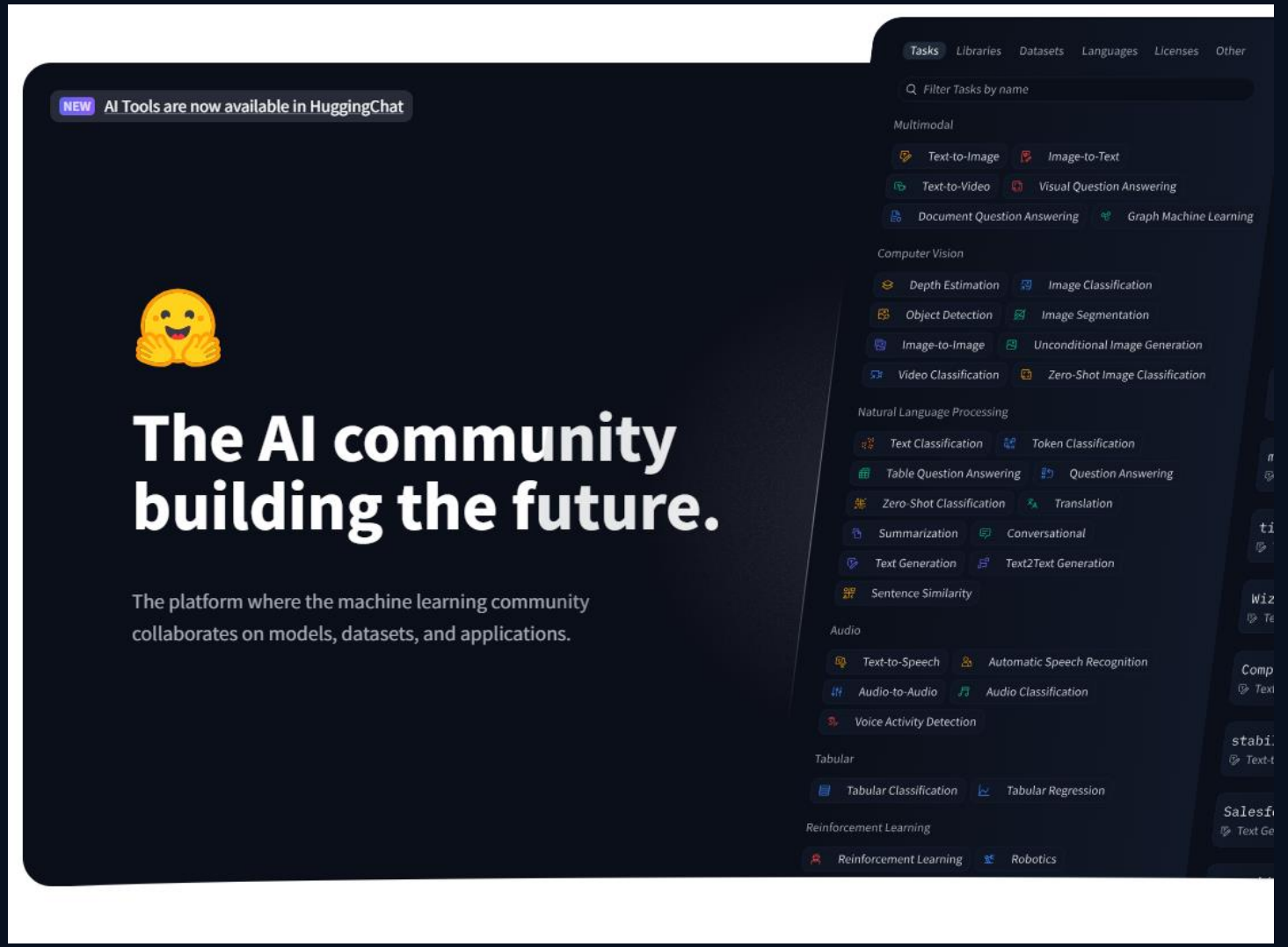
As language models, LLMs acquire these abilities by learning statistical relationships from vast amounts of text during a self-supervised and semi-supervised training process.

LLM

HuggingFace

The platform where the machine learning community collaborates on models, datasets, and applications.

[Hugging Face – The AI community building the future.]



Download and Deploy the LLM

- We use Phi-3-mini as a lightweight LLM
- We download it from HuggingFace
- We can host it locally with:
  - OLLAMA
  - LlamaEdge

Download and Deploy the LLM with LlamaEdge

Getting Started - Generative AI with Phi-3-mini: A Guide to Inference and Deployment (microsoft.com)

Download wasmedge

```
curl -sSf https://raw.githubusercontent.com/WasmEdge/WasmEdge/master/utils/install.sh | bash -s -- --plugin wasi_nn-ggml
```

download lastest version of the llama api server

```
curl -LO https://github.com/LlamaEdge/LlamaEdge/releases/latest/download/llama-api-server.wasm
```

download small test application

```
curl -LO https://github.com/LlamaEdge/chatbot-ui/releases/latest/download/chatbot-ui.tar.gz

tar xzf chatbot-ui.tar.gz
```

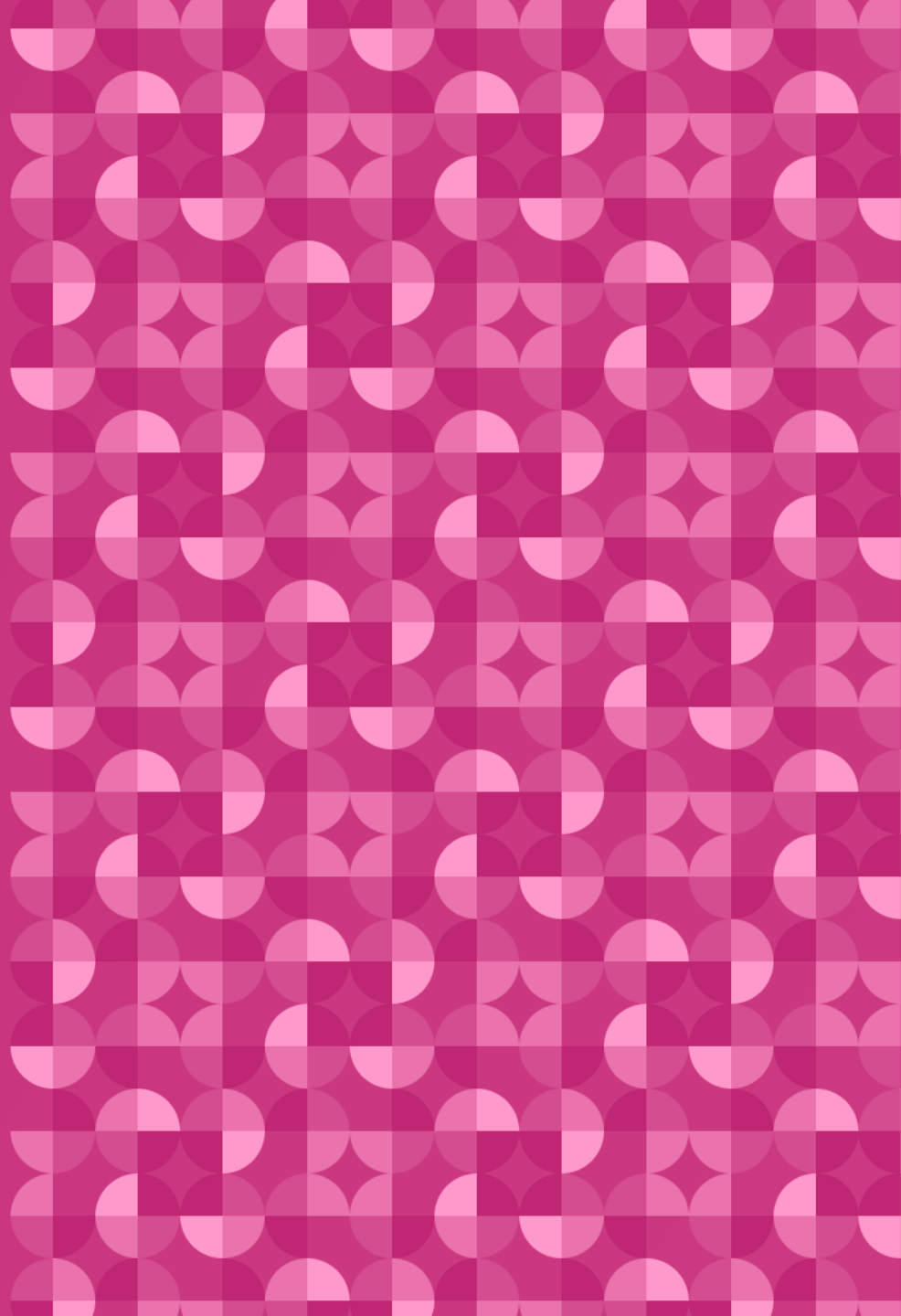
2

Sematinc Kernel

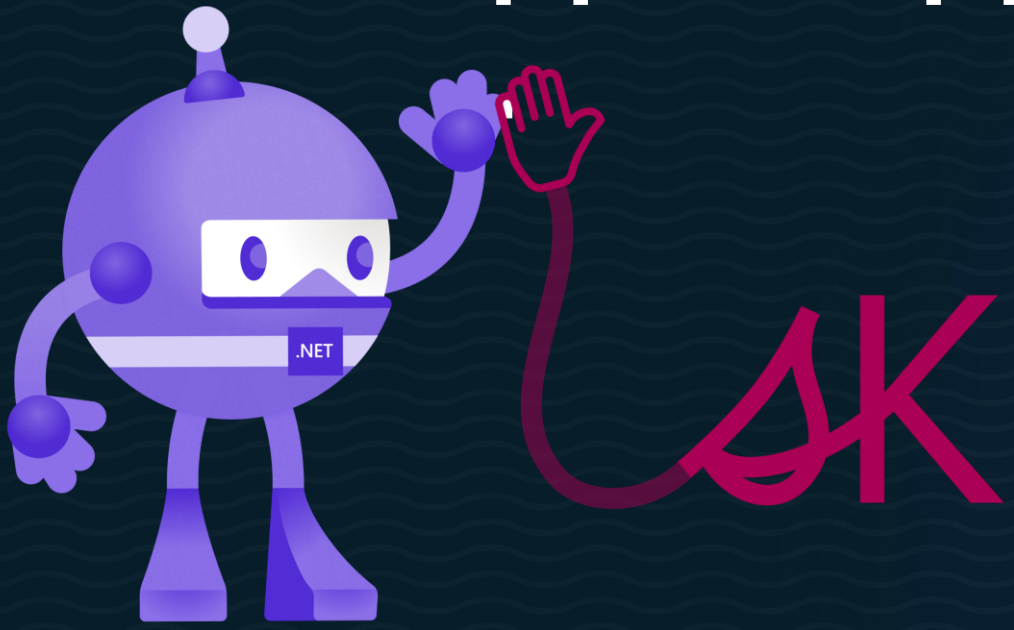
- Short Semantic Kernel Intro

Unleashing AI Superpowers: Task Automation with Semantic Kernel

With the aid of plugins, planners, and personas.



So SK wants to support app teams.



What is Semantic Kernel?

Semantic Kernel is a lightweight open-source SDK that lets you orchestrate native code with LLMs.

Semantic Kernel

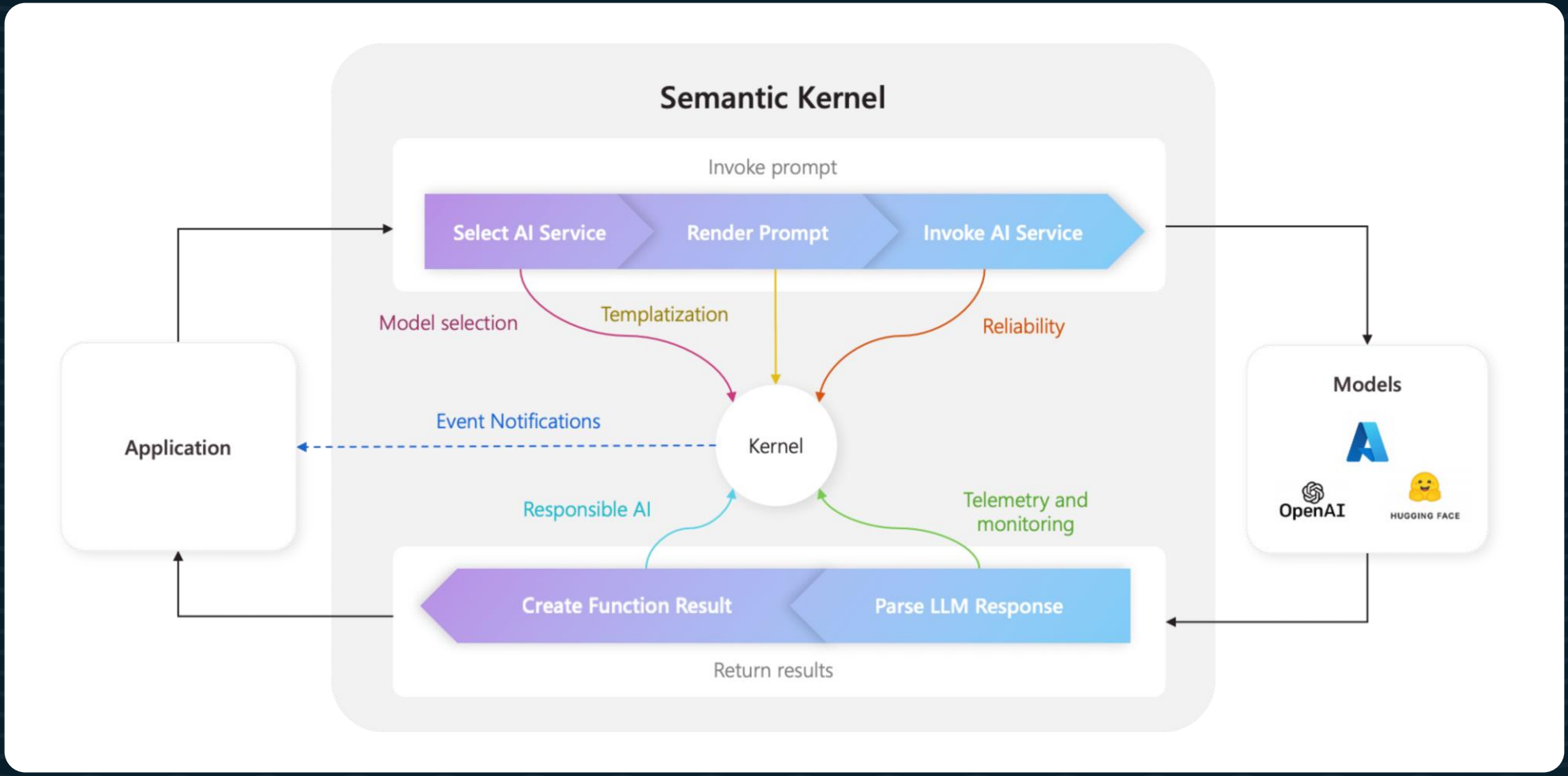
Semantic Kernel is available in...

C# (V1.10)

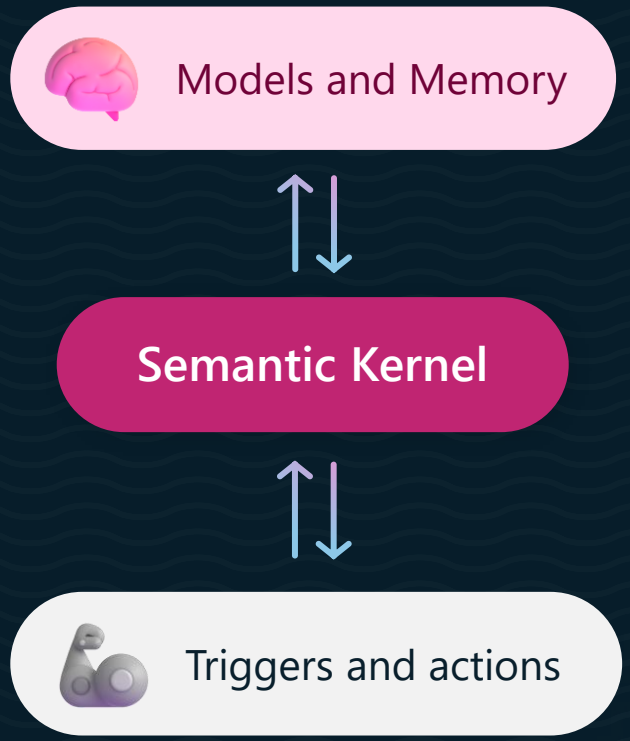
Python (Beta)

Java (Release Candidate)

The kernel is at the center of it all



Semantic Kernel is extensible



What is LangChain?

README | Code of conduct | MIT license | Security

🦜🔗 LangChain

⚡ Build context-aware reasoning applications ⚡

release v0.1.7 | CI failing | downloads/month 6M | License MIT | Follow @LangChainAI | LangChain Community

Dev Containers Open | Open in GitHub Codespaces | Stars 77k | Dependency Status | open issues 1.4k

Looking for the JS/TS library? Check out LangChain.js.

To help you ship LangChain apps to production faster, check out LangSmith. LangSmith is a unified developer platform for building, testing, and monitoring LLM applications. Fill out this form to speak with our sales team.

Quick Install

With pip:

```
pip install langchain
```

With conda:

```
conda install langchain -c conda-forge
```

Supported languages

- ❑ Python (0.1.7)
- ❑ TypeScript (0.1.18)
- ❑ Java (0.27.1)

At the core of LangChain are chains

```python
from langchain_core.runnables import RunnablePassthrough


prompt = ChatPromptTemplate.from_template(
    "Tell me a short joke about {topic}"
)
output_parser = StrOutputParser()
model = ChatOpenAI(model="gpt-3.5-turbo")
chain = (
    {"topic": RunnablePassthrough()}
    | prompt
    | model
    | output_parser
)

chain.invoke("ice cream")
```

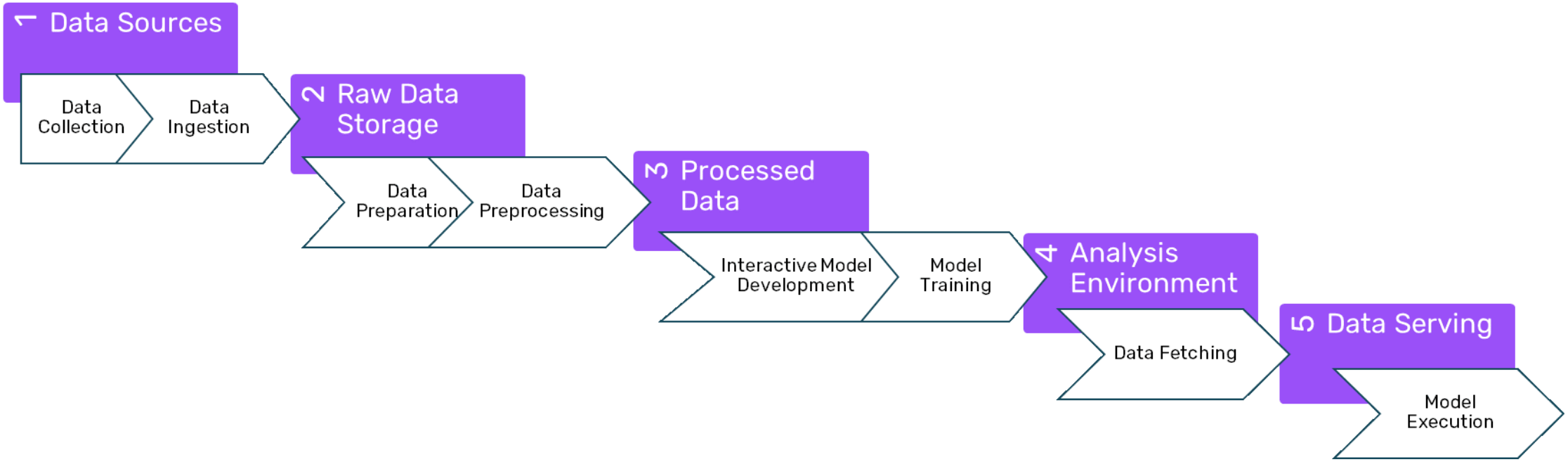
With chains, you pipe together elements (e.g., prompts, models, and parsers) before invoking it

Supported languages

- ❑ Python (0.1.7)
- ❑ TypeScript (0.1.18)
- ❑ Java (0.27.1)

LangChain

LangChain was built by and for AI researchers who have experience building AI data pipelines.



Chains are familiar to them!

How do you actually develop with Semantic Kernel?

It all starts with understanding 3 key things...

1. Plugins
2. Planners
3. Personas

1

Plugins

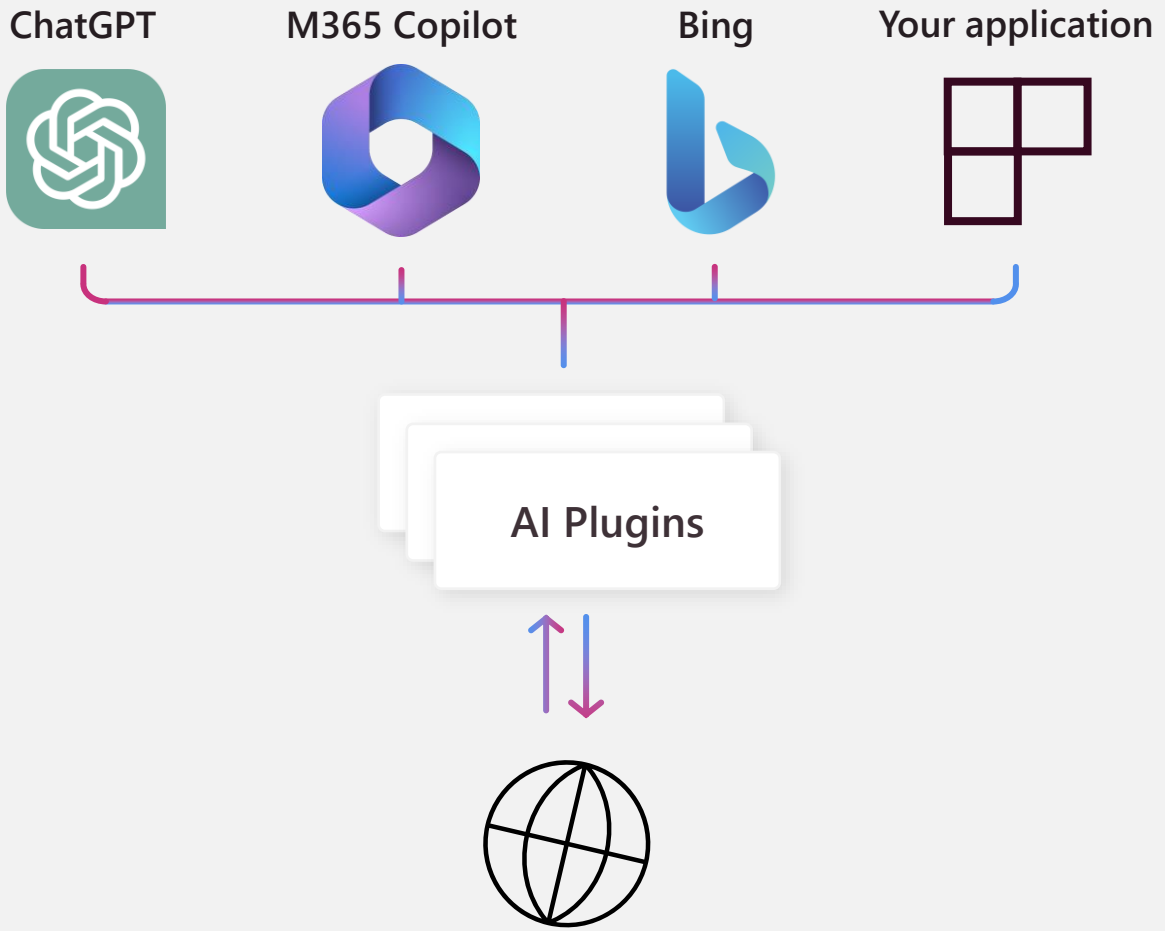
The core building block for Copilots and AI driven applications.

Connect AI to the real world

What *is* a plugin?

Chatbots are *nice*, but they aren't *useful* to your users until they can interact with the real world by...

1. Retrieve data
2. Sending emails
3. Completing sales
4. Making orders
5. And more!



2

Planners

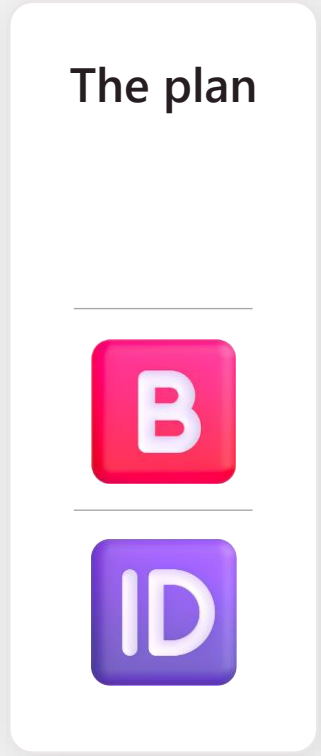
The magic that combines plugins together to accomplish a user's goal.

Orchestrating plugins

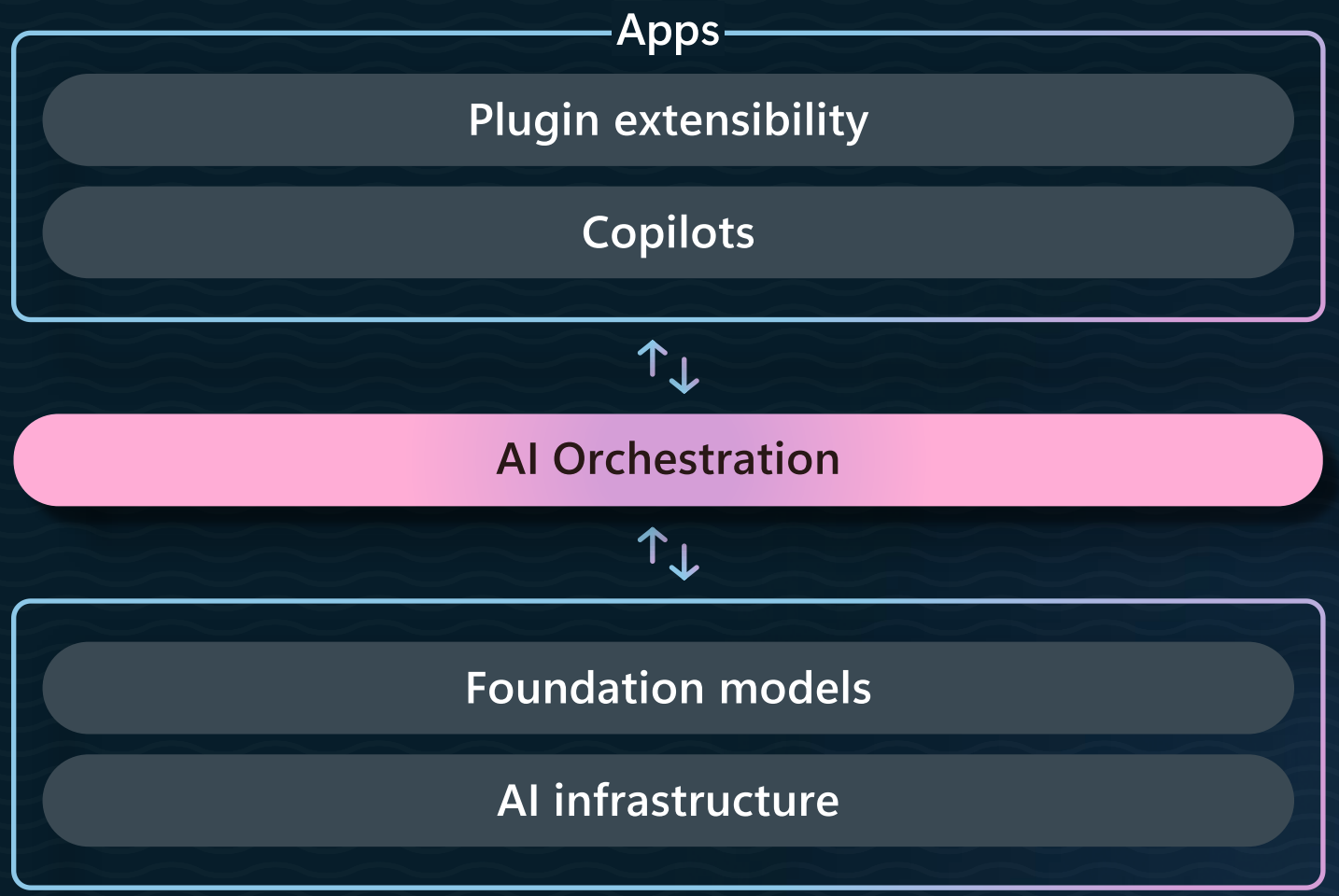
At the core of every copilot is a planner

Planners tell a Copilot what it should do next with the tools it has. Semantic Kernel has 5 generations of planners

1. Action planner
2. Sequential planner
3. Stepwise planner
* Automatic function calling
4. Handlebars planner
5. Stepwise v2 planner



With a planner, you can build your own AI apps



1. Plugins
2. Planners
3. Personas

3

Personas

Creating multi-agent experiences with Semantic Kernel.

Recap

Personas are just instructions+plugins

By combining a system prompt (i.e., instructions) with a set of plugins, you can recreate a “persona” that can complete user requests.

This mimics how “personas” in experience design have roles and tools to complete jobs-to-be-dones.

Also called agents, copilots, GPTs, and assistants.

Coder agent

Instructions

You are a python coder that can create classes and functions that address the user’s needs.

Plugins

Python plugin

Doc plugin

Agents

With function calling, you already have an "agent”, but it’d be helpful to have an abstraction for this…

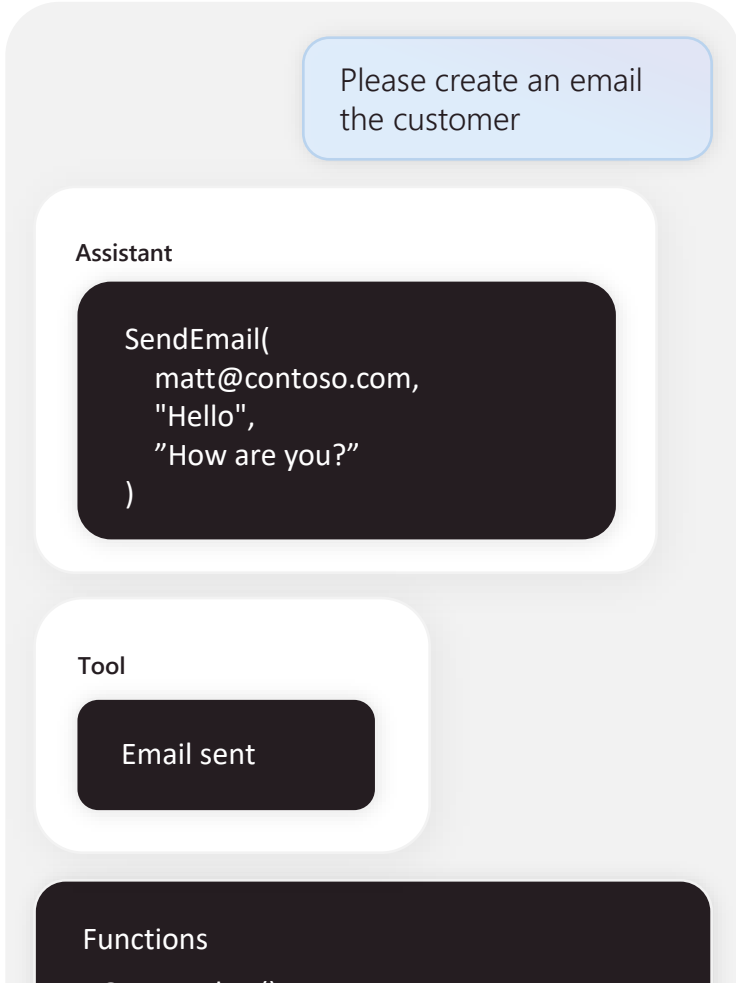
```
// Start the conversation
while (true)
{
    // Get user input
    Console.Write("User > ");
    history.AddUserMessage(Console.ReadLine()!);

    // Enable auto function calling
    OpenAIPromptExecutionSettings openAIPromptExecutionSettings = new()
    {
        ToolCallBehavior = ToolCallBehavior.AutoInvokeKernelFunctions
    };

    // Stream the response from the AI
    var result = chatCompletionService.GetStreamingChatMessageContentsAsync(
        history,
        executionSettings: openAIPromptExecutionSettings,
        kernel: kernel);

    // Print the results
    string finalResult = "";
    Console.Write("Assistant > ");
    await foreach (var chunk in result)
    {
        Console.Write(chunk.Content);
        finalResult += chunk.Content;
    }
    Console.WriteLine();

    // Add the message from the agent to the chat history
    history.AddMessage(AuthorRole.Assistant, finalResult);
}
```



Agents

OpenAI introduced the Assistants API to address this need

Assistants API Beta

The Assistants API allows you to build AI assistants within your own applications. An Assistant has instructions and can leverage models, tools, and files to respond to user queries. The Assistants API currently supports three types of tools: Code Interpreter, File Search, and Function calling.

You can explore the capabilities of the Assistants API using the Assistants playground or by building a step-by-step integration outlined in this guide.

Overview

A typical integration of the Assistants API has the following flow:

1. Create an Assistant by defining its custom instructions and picking a model. If helpful, add files and enable tools like Code Interpreter, File Search, and Function calling.
2. Create a Thread when a user starts a conversation.
3. Add Messages to the Thread as the user asks questions.
4. Run the Assistant on the Thread to generate a response by calling the model and the tools.

This starter guide walks through the key steps to create and run an Assistant that uses Code Interpreter. In this example, we're creating an Assistant that is a personal math tutor, with the Code Interpreter tool enabled.

Other benefits of the API include...

1. Managed threads
2. Code Interpreter tool
3. Retrieval tool

3

Demo SK

- Run some basic examples