Semantic Kernel

Where simplicity meets productive AI

Unleashing the Power of AI
Beginner to Intermediate

Roger Barreto

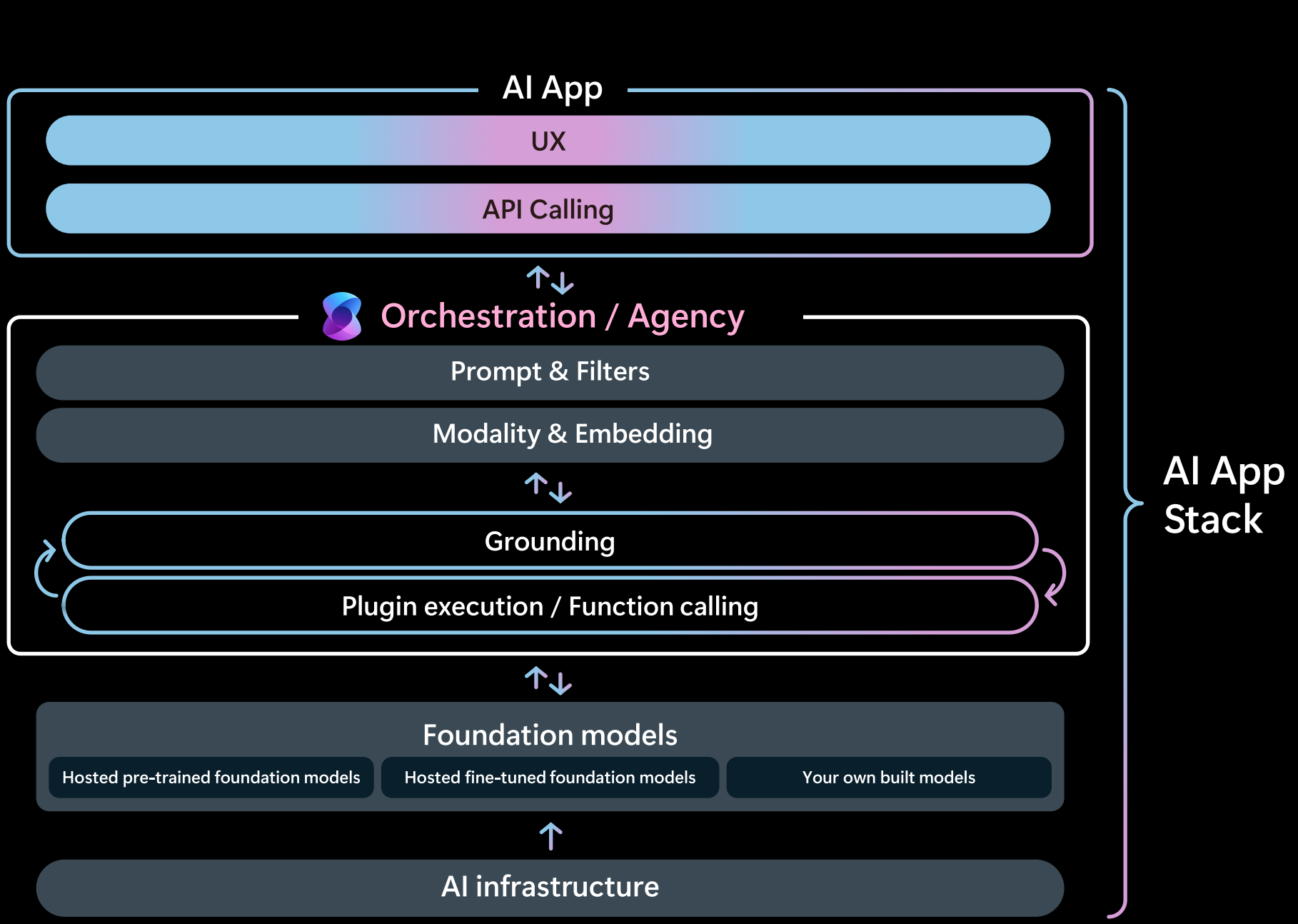
Sr. Software Engineer

CoreAI

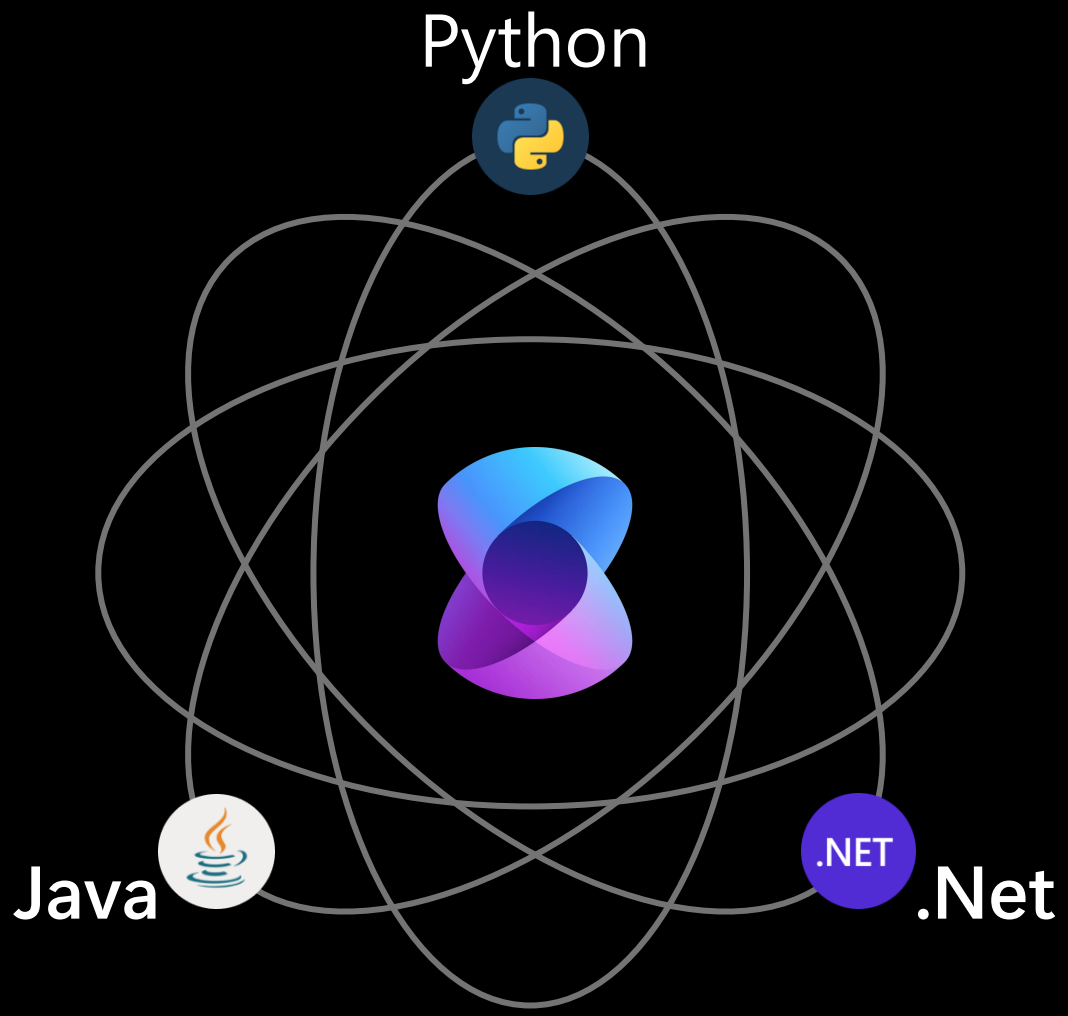
RogerBarreto

RogerBarreto

RogerBarretto


AI App
UX
API Calling
Orchestration / Agency
Prompt & Filters
Modality & Embedding
Grounding
Plugin execution / Function calling
Foundation models
Hosted pre-trained foundation models
Hosted fine-tuned foundation models
Your own built models
AI infrastructure
AI App Stack

Semantic Kernel is a lightweight open-source orchestration SDK middleware that lets you easily integrate AI in your apps.



Our Product Pillars

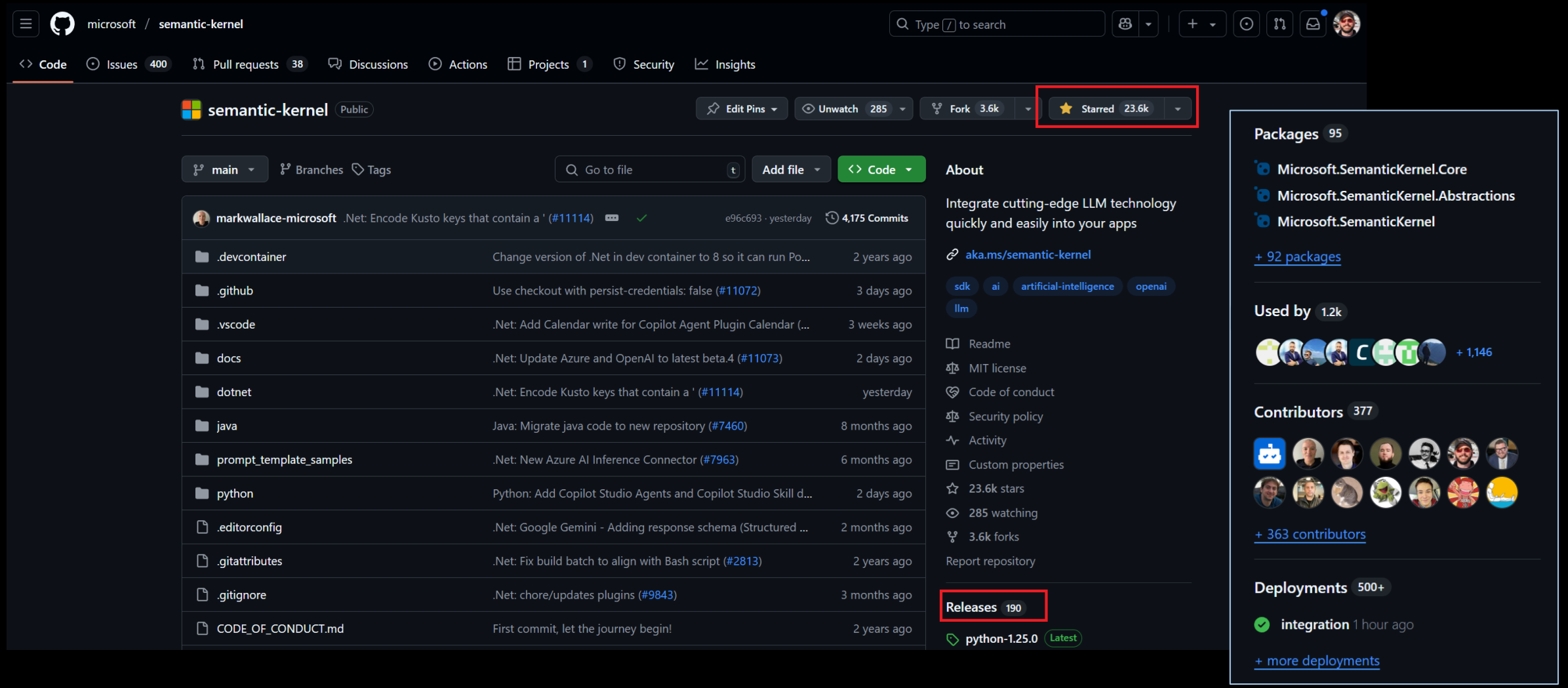
Open source
Reliability
Performance
Trustworthy
Extensible

Our Product Pillars

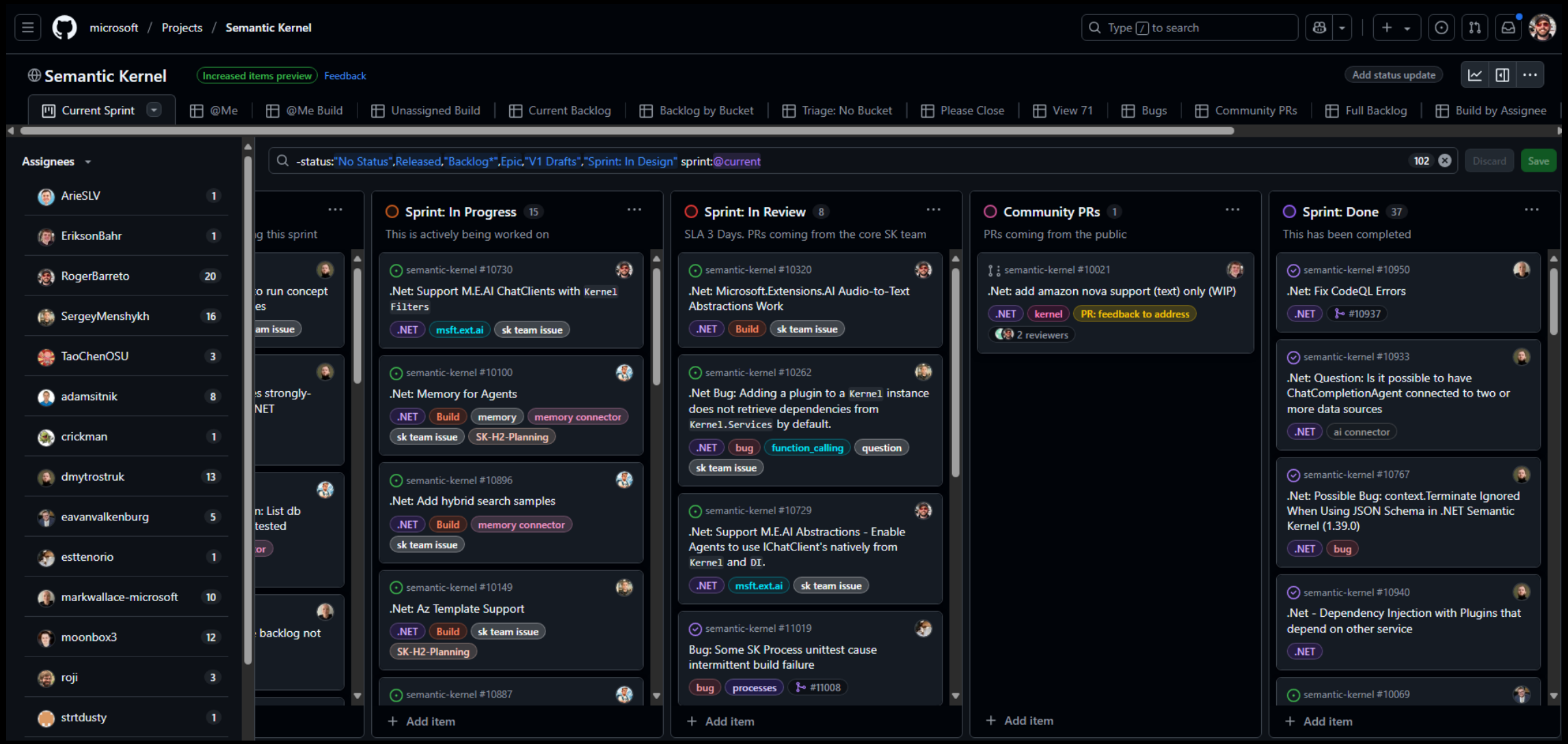
Open source

- MIT License
- Visibility & Transparency
- Open for contribution
- Quick learning curve
- Forks & Mods

GitHub Repository



GitHub Project



Our Product Pillars

Performance

- Fast & Lightweight
- Simple Async APIs
- Optimized execution paths
- Scales to millions of users
- Production-ready

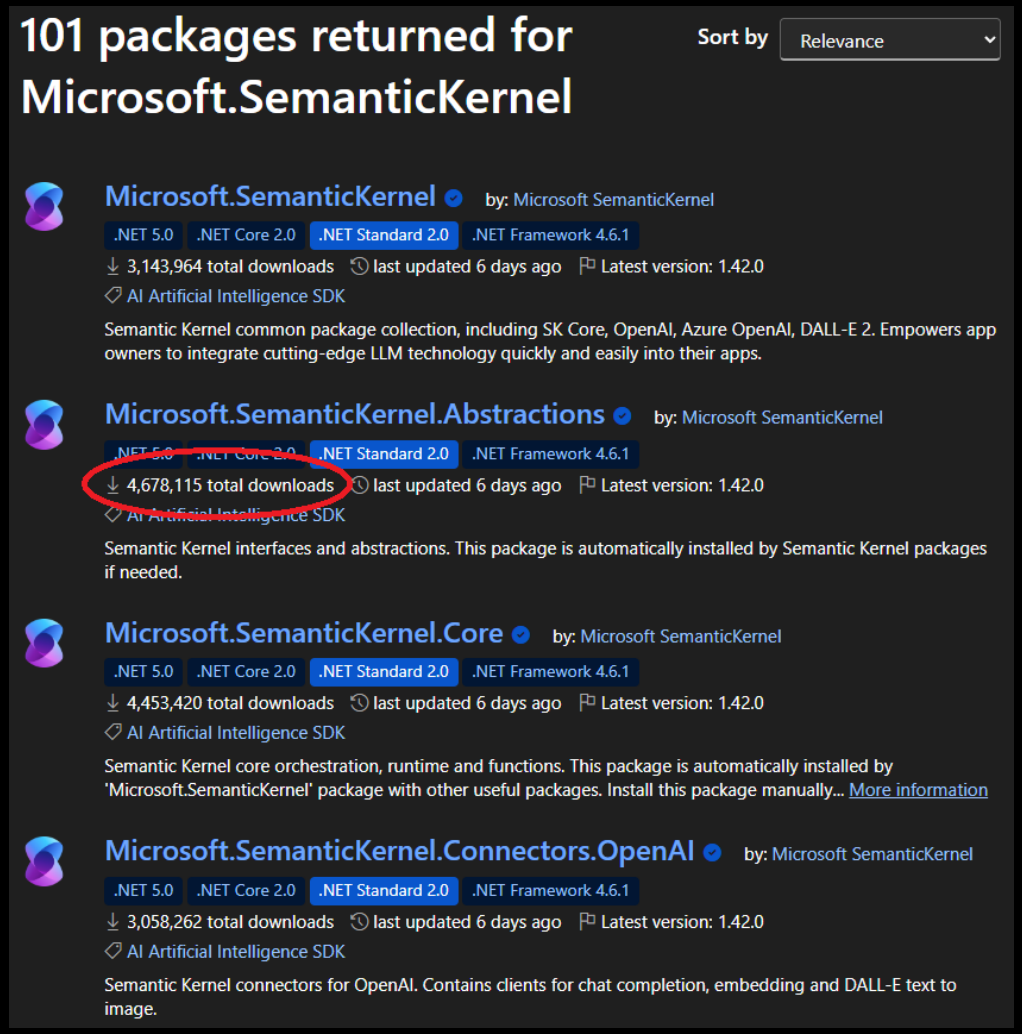
Our Product Pillars

Reliability

- Modular packages
- Telemetry & Logging enabled
- Dependency Injection enabled
- Break-glass flexibility
- Versioning stability

Packages Usage & Adoption

.Net: https://www.nuget.org/packages?q=Microsoft.SemanticKernel



Python: https://pypistats.org/packages/semantic-kernel

Our Product Pillars

Trustworthy

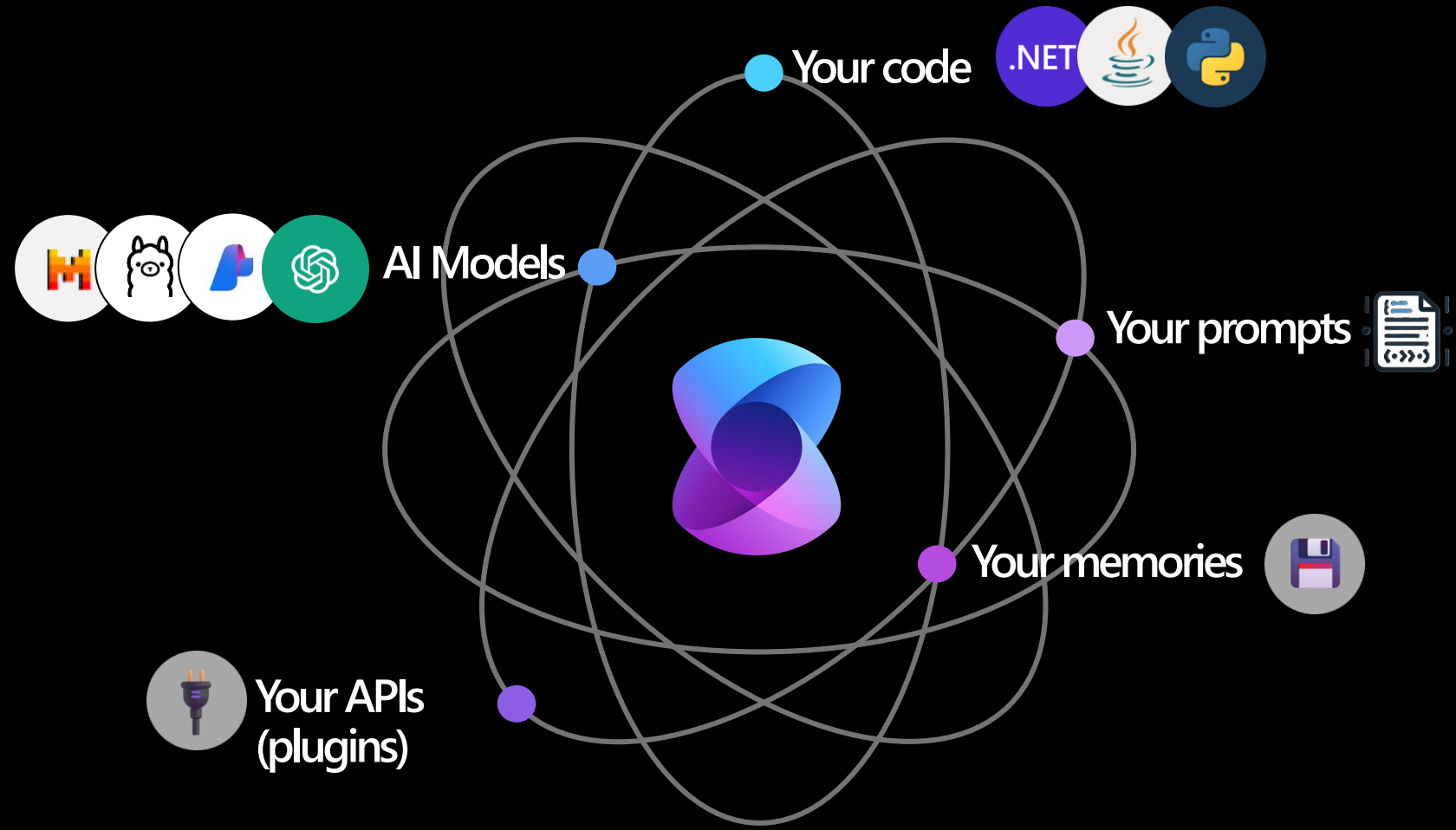
- Security checks
- Code coverage checks
- Code review and quality
- Rigor at Testing
- Feature discussion and open architecture records (ADRs)
- Minimum of 2 core team approvals

Our Product Pillars

Extensibility

- Built on top of abstractions
- Create your own customized SK component
- Multiple providers (and growing)

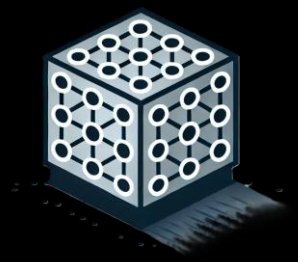
Designed for modular extensibility



Main components (AI Services)

AI Services (Connectors)

Vector Databases



AI Models

Main components (Plugins)

Your C# Classes

Open API Spec

Disk Files

Native Code

Plugins (Functions)

AI Models

Prompt Template Engines

Main components (Prompt Templates)

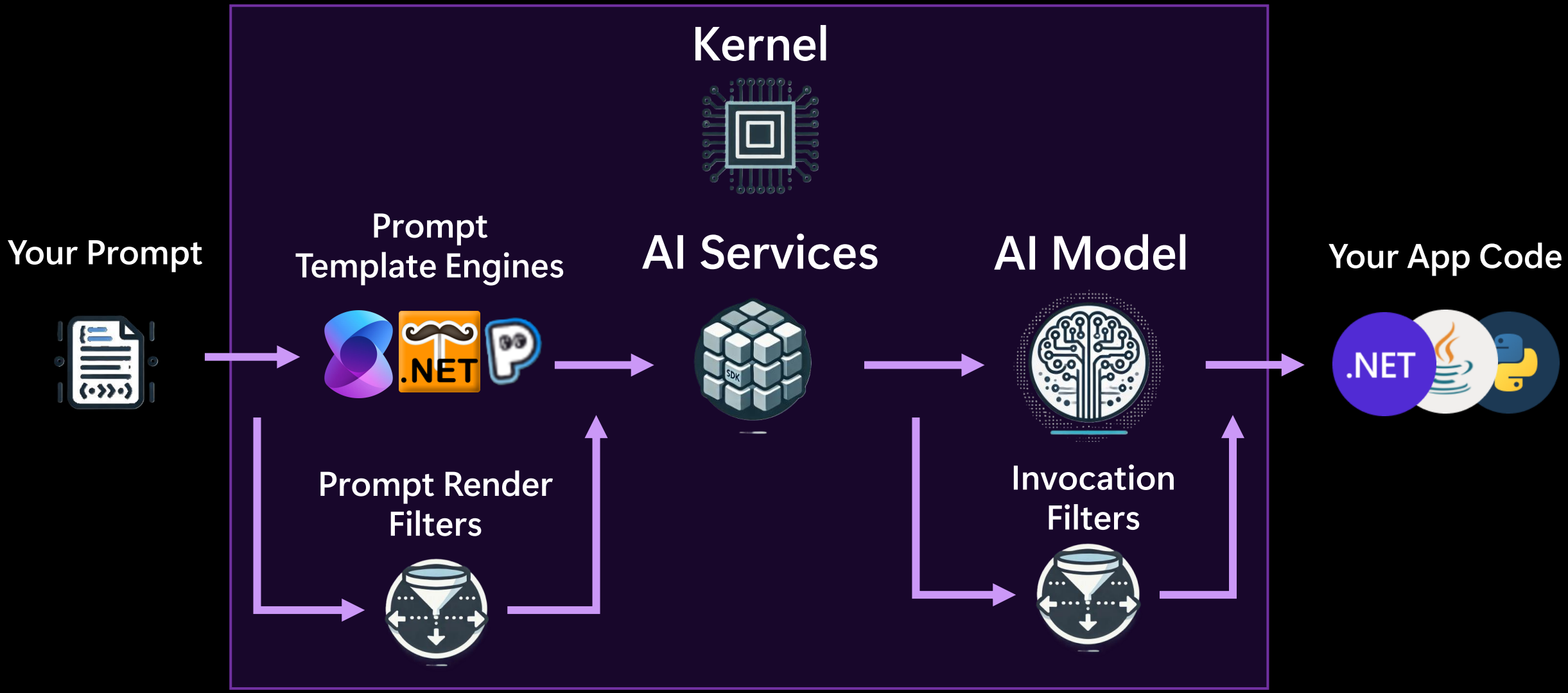
Your Prompt

Prompt
Template Engine

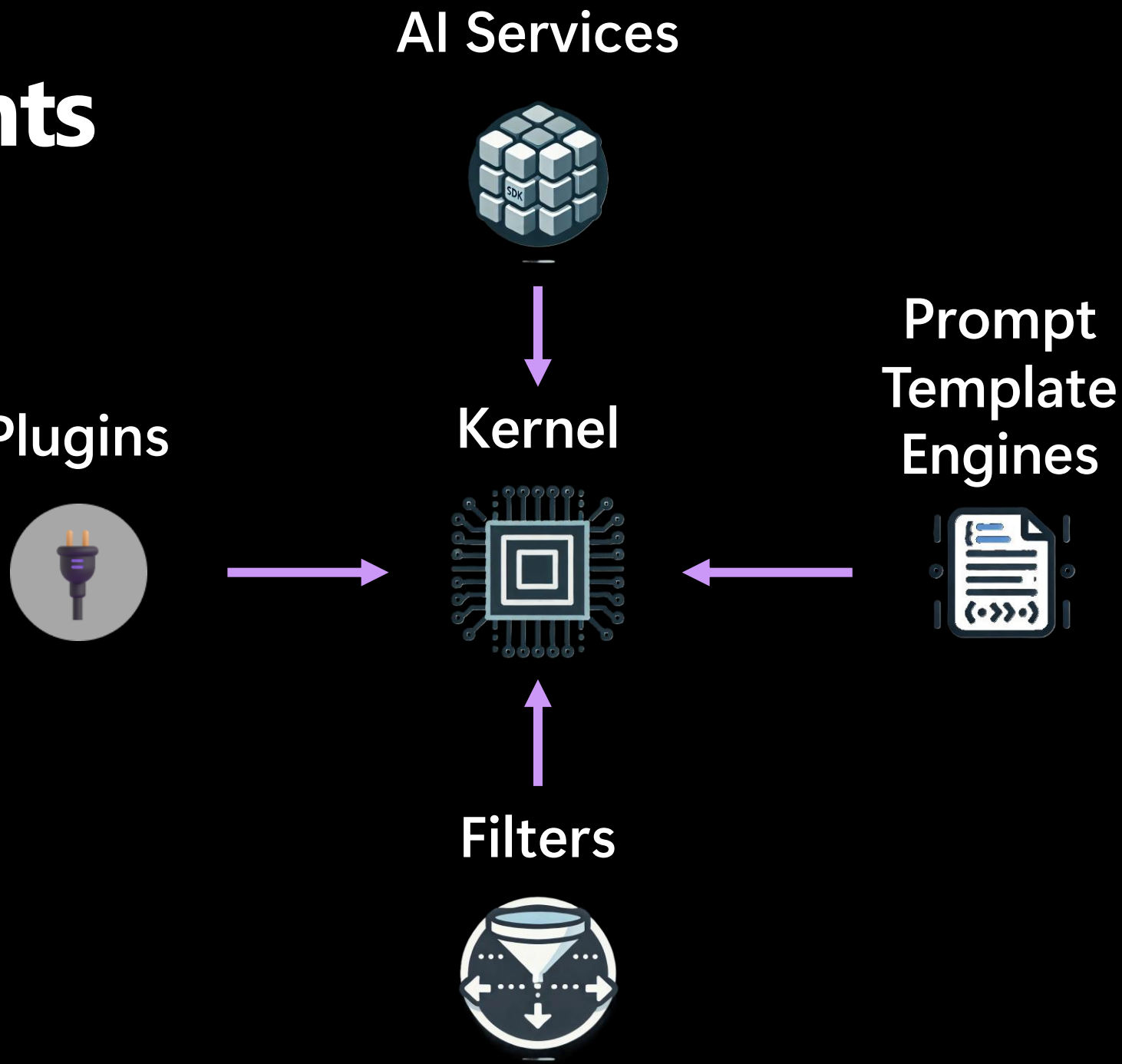
AI Service

AI Models

Main components (Kernel Filters)



Main components (Kernel)

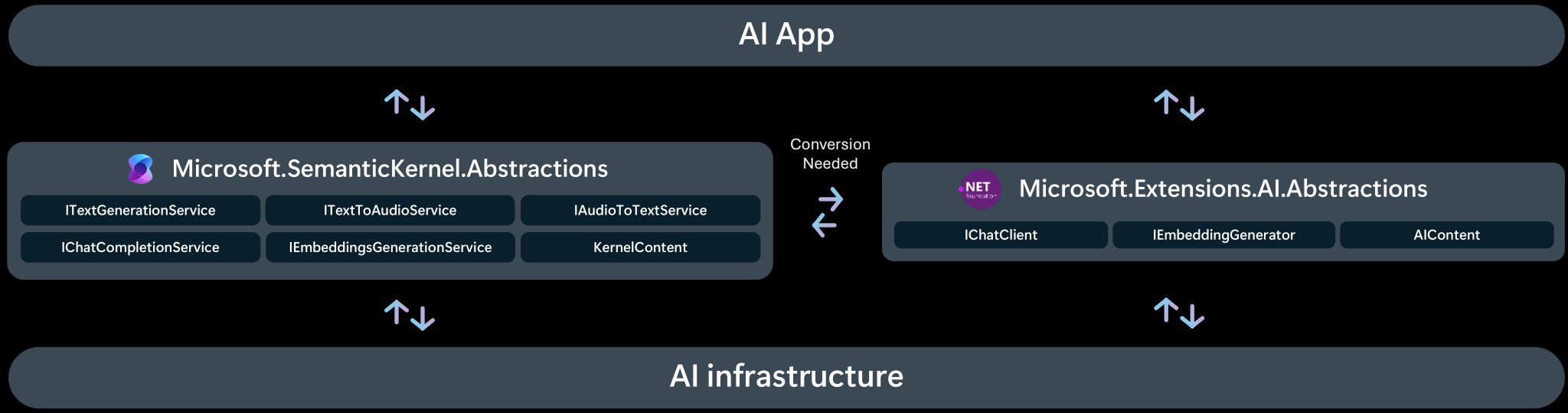


Additional components

- AI Service Selector
- Prompt Template Engine
- Content Abstractions (Chat Messages, Image, Audio, Binary)
- Service Modalities (Text Generation, Chat Completion, Text-to-Image, etc)
- Vector Data & Text Search (Embeddings + Search)
- Microsoft Extensions AI Types

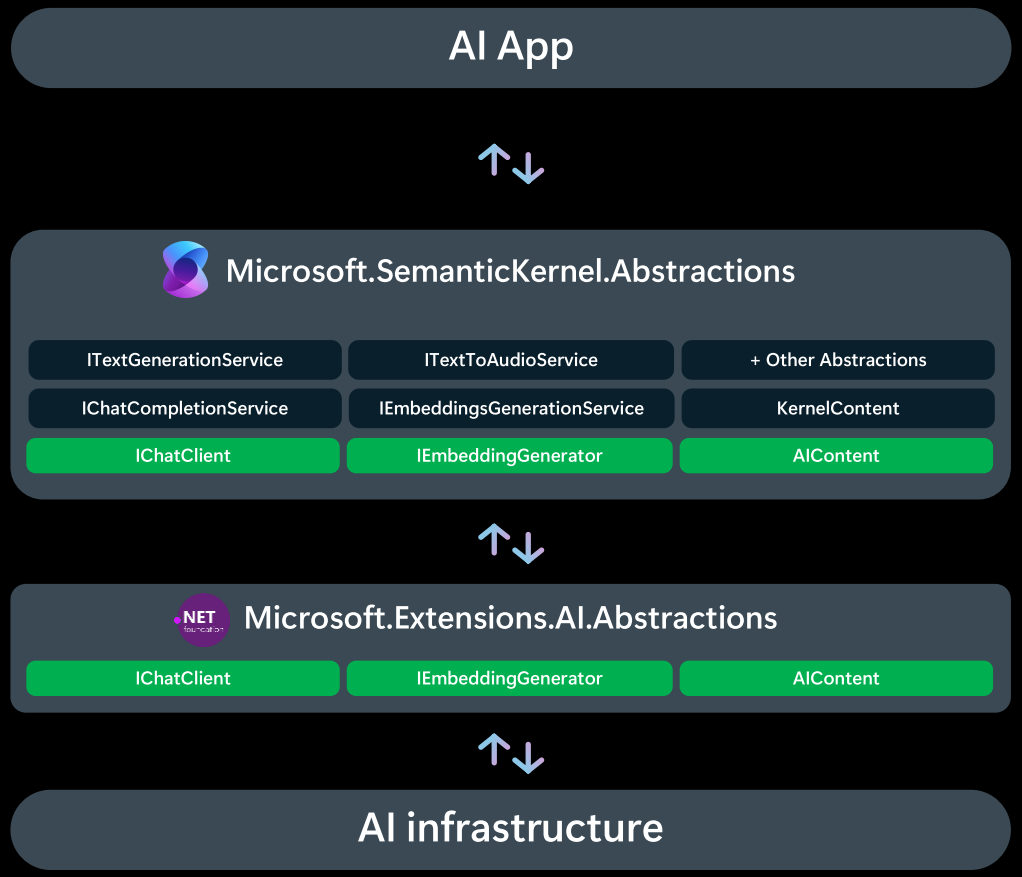
Semantic Kernel + Microsoft.Extensions.AI (Preview)

Extensions project was released in last October as the foundation abstraction for AI types and services.
In case you already have M.E.AI code you can use it with Semantic Kernel simply thru our conversion extensions and vice-versa.



https://devblogs.microsoft.com/dotnet/introducing-microsoft-extensions-ai-preview/

Semantic Kernel + Microsoft.Extensions.AI (GA)



Near term, when M.E.AI becomes GA

SK will have native Support for M.E.AI Types

This change moves the M.E.AI abstractions as the foundation abstraction layer that semantic kernel can rely and use natively.

Coding time!

github.com/RogerBarreto/semantic-kernel-demos

README | MIT license

Semantic Kernel Demonstrations

This repository contains a collection of Semantic Kernel demonstration projects. Semantic Kernel is a powerful framework for building AI applications using Large Language Models (LLMs) and other AI models. These demonstrations showcase various use cases and features of Semantic Kernel, providing insights into its capabilities and potential applications.

Beginner Hands-On Semantic Kernel Demos

A collection of .NET samples demonstrating various Semantic Kernel features:

1. Kernel Prompting: Basic prompt handling and streaming
2. Kernel Dependency Injection: Using DI with Semantic Kernel
3. Kernel Prompt Template: Working with prompt templates and variables
4. Chat Service with Chat History: Managing conversation history
5. Prompt Execution Settings: Configuring prompt execution
6. Grounding Prompts with Plugins: Using plugins for context
7. Plugin Function Calling: Implementing and using plugins
8. Using OpenAPI Plugins: Integrating external APIs via OpenAPI
9. Using Kernel Filters: Implementing kernel filters
10. Aspire Dashboard + OpenTelemetry: Monitoring and telemetry
11. Multiple AI Models: Working with different AI providers
12. Multi-Modality: Handling text, images, and audio

1. Kernel Prompting (Simplicity)

```
using Microsoft.Extensions.Configuration;
using Microsoft.SemanticKernel;

Console.WriteLine("=== Kernel prompting  - Non-Streaming ===\n");

var apiKey = new ConfigurationBuilder()
    .AddUserSecrets<Program>()
    .Build()["OpenAI:ApiKey"]!;

var kernel = Kernel.CreateBuilder()
    .AddOpenAIChatCompletion("gpt-4o", apiKey)
    .Build();
var prompt = "Why is Neptune blue?";

Console.WriteLine(await kernel.InvokePromptAsync(prompt));

Console.WriteLine("\n=== Kernel prompting - Streaming ===\n");
await foreach (var token in kernel.InvokePromptStreamingAsync(prompt))
{
    Console.Write(token);
}
```

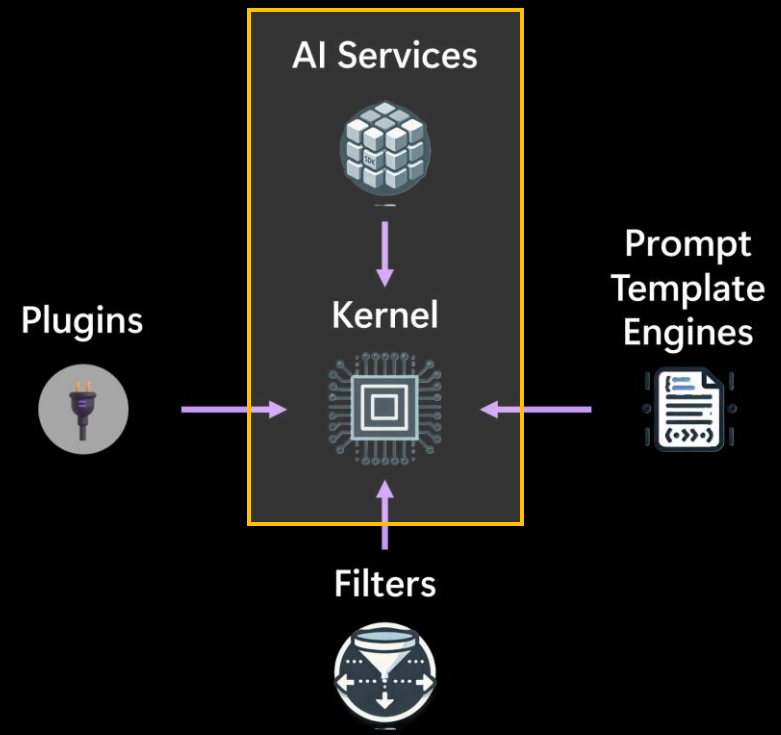
Our Product Pillars

Performance

- Fast & Lightweight
- Simple Async APIs
- Optimized execution paths
- Scales to millions of users
- Production-ready

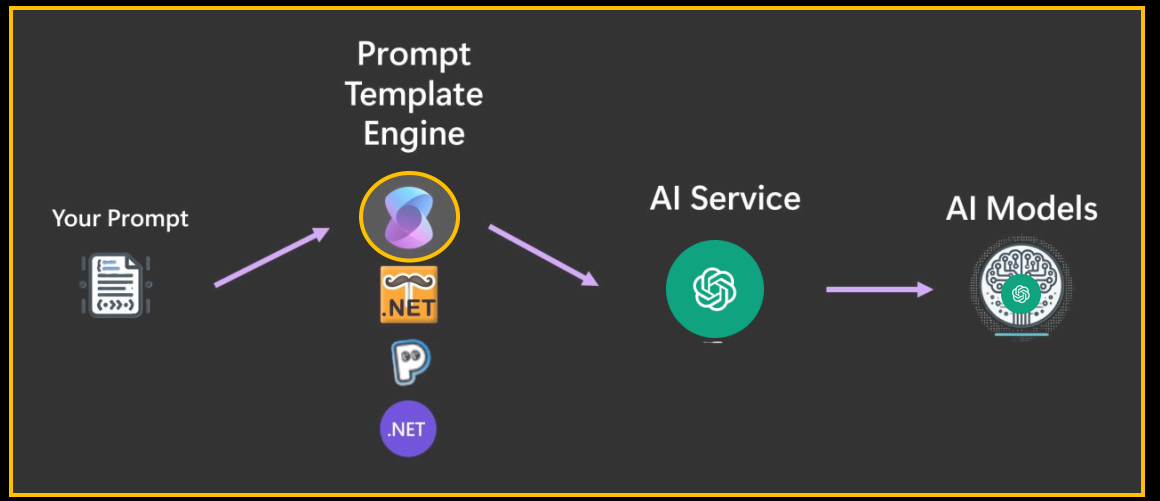
1. Kernel Prompting (AI Service)

```
1  using Microsoft.Extensions.Configuration;
2  using Microsoft.SemanticKernel;
3
4  Console.WriteLine("=== Kernel prompting  - Non-Streaming ===\n");
5
6  var apiKey = new ConfigurationBuilder()
7      .AddUserSecrets<Program>()
8      .Build()["OpenAI:ApiKey"]!;
9
10 var kernel = Kernel.CreateBuilder()
11     .AddOpenAIChatCompletion("gpt-4o", apiKey)
12     .Build();
13 var prompt = "Why is Neptune blue?";
14
15 Console.WriteLine(await kernel.InvokePromptAsync(prompt));
16
17 Console.WriteLine("\n=== Kernel prompting - Streaming ===\n");
18 await foreach (var token in kernel.InvokePromptStreamingAsync(prompt))
19 {
20     Console.Write(token);
21 }
```



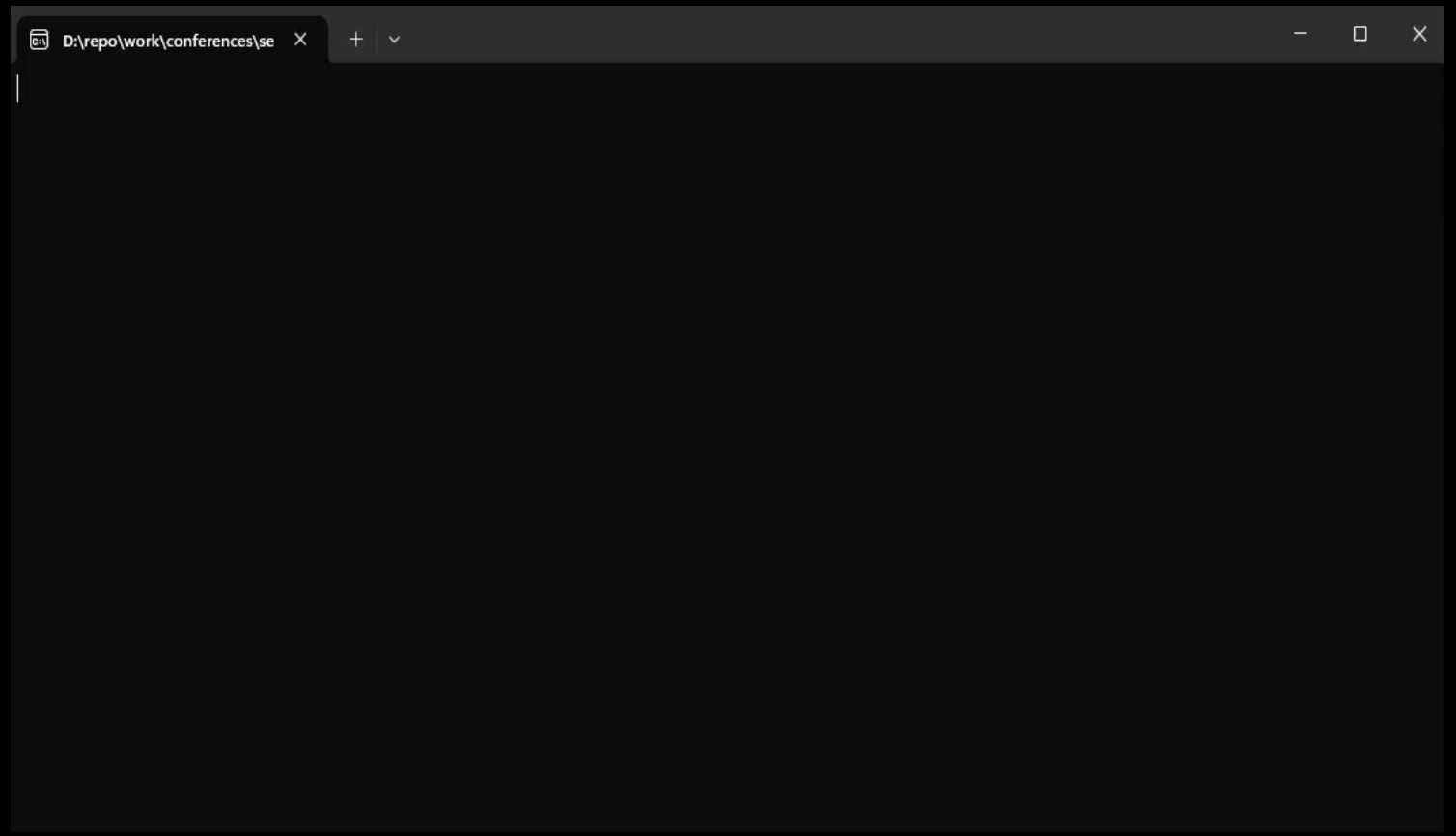
1. Kernel Prompting (Invocation)

```
1   using Microsoft.Extensions.Configuration;
2   using Microsoft.SemanticKernel;
3
4   Console.WriteLine("=== Kernel prompting  - Non-Streaming ===\n");
5
6   var apiKey = new ConfigurationBuilder()
7       .AddUserSecrets<Program>()
8       .Build()["OpenAI:ApiKey"]!;
9
10  var kernel = Kernel.CreateBuilder()
11      .AddOpenAIChatCompletion("gpt-4o", apiKey)
12      .Build();
13  var prompt = "Why is Neptune blue?";
14
15  Console.WriteLine(await kernel.InvokePromptAsync(prompt));
16
17  Console.WriteLine("\n=== Kernel prompting - Streaming ===\n");
18  await foreach (var token in kernel.InvokePromptStreamingAsync(prompt))
19  {
20      Console.Write(token);
21  }
```



1. Kernel Prompting (Execution)

```
1  using Microsoft.Extensions.Configuration;
2  using Microsoft.SemanticKernel;
3
4  Console.WriteLine("=== Kernel prompting  - Non-Streaming ===\n");
5
6  var apiKey = new ConfigurationBuilder()
7      .AddUserSecrets<Program>()
8      .Build()["OpenAI:ApiKey"]!;
9
10 var kernel = Kernel.CreateBuilder()
11     .AddOpenAIChatCompletion("gpt-4o", apiKey)
12     .Build();
13 var prompt = "Why is Neptune blue?";
14
15 Console.WriteLine(await kernel.InvokePromptAsync(prompt));
16
17 Console.WriteLine("\n=== Kernel prompting - Streaming ===\n");
18 await foreach (var token in kernel.InvokePromptStreamingAsync(prompt))
19 {
20     Console.Write(token);
21 }
```



2. Kernel Dependency Injection

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.SemanticKernel;

#pragma warning disable SKEXP0070 // Type is for evaluation purposes on

Console.WriteLine("=== Kernel Dependency Injection ===\n");

var modelId = "llama3.2";
var endpoint = new Uri("http://localhost:11434");

var services = new ServiceCollection();
services.AddOllamaChatCompletion(modelId, endpoint);
services.AddKernel();

var serviceProvider = services.BuildServiceProvider();
var kernel = serviceProvider.GetRequiredService<Kernel>();

var prompt = "Why is Neptune blue?";
Console.WriteLine($"User > {prompt}");
Console.Write("Assistant > ");
await foreach (var token in kernel.InvokePromptStreamingAsync(prompt))
{
    Console.Write(token);
};
```

Our Product Pillars

Reliability

- Modular packages
- Telemetry & Logging enabled
- Dependency Injection enabled
- Break-glass flexibility
- Versioning stability

2. Kernel Dependency Injection

```
1   using Microsoft.Extensions.DependencyInjection;
2   using Microsoft.SemanticKernel;
3
4   #pragma warning disable SKEXP0070 // Type is for evaluation purposes onl
5
6   Console.WriteLine("=== Kernel Dependency Injection ===\n");
7
8   var modelId = "llama3.2";
9   var endpoint = new Uri("http://localhost:11434");
10
11  var services = new ServiceCollection();
12  services.AddOllamaChatCompletion(modelId, endpoint);
13  services.AddKernel();
14
15  var serviceProvider = services.BuildServiceProvider();
16  var kernel = serviceProvider.GetRequiredService<Kernel>();
17
18  var prompt = "Why is Neptune blue?";
19  Console.WriteLine($"User > {prompt}");
20  Console.Write("Assistant > ");
21  await foreach (var token in kernel.InvokePromptStreamingAsync(prompt))
22  {
23      Console.Write(token);
24  };
```

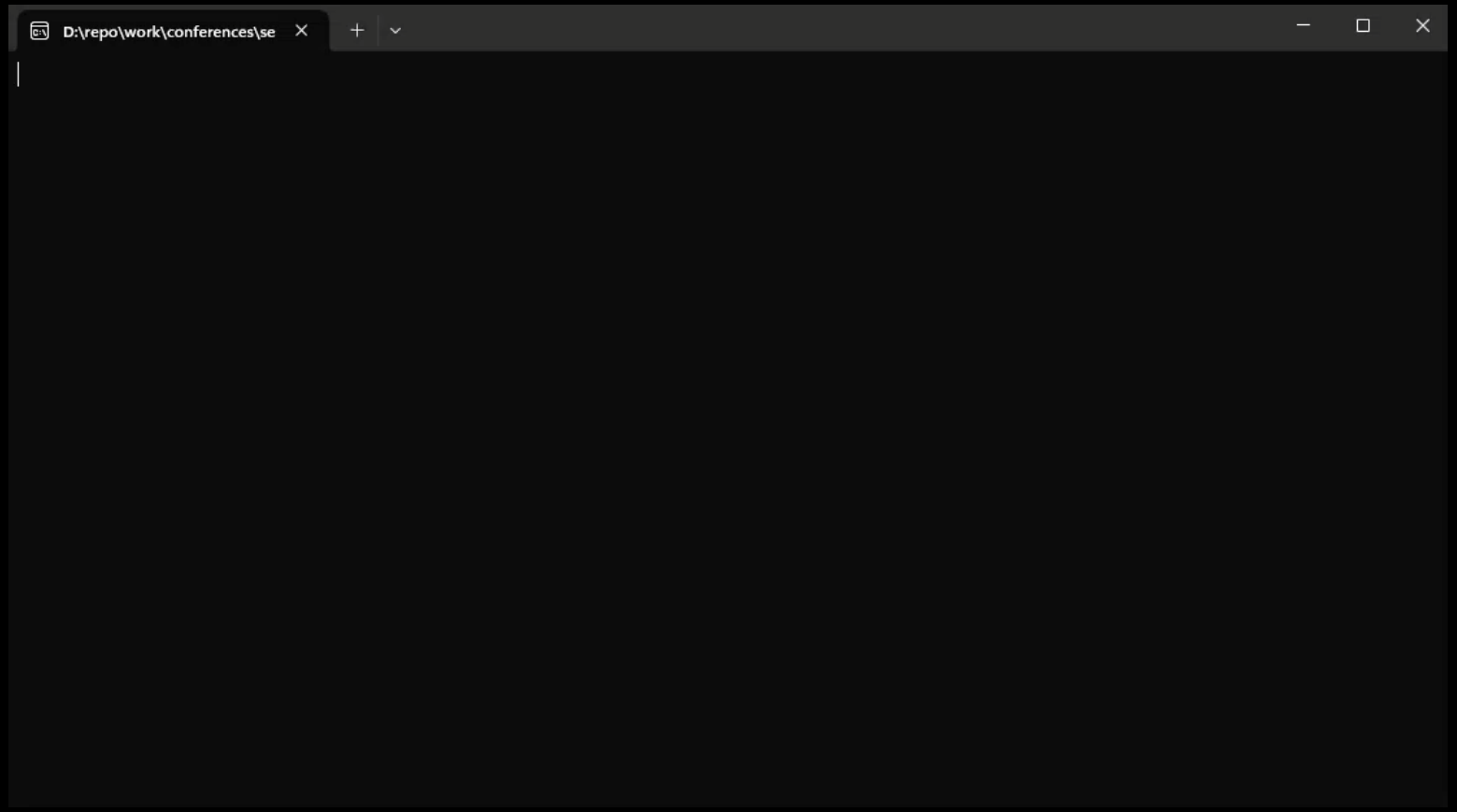
Our Product Pillars

Reliability

- Modular packages
- Telemetry & Logging enabled
- **Dependency Injection enabled**
- Break-glass flexibility
- Versioning stability

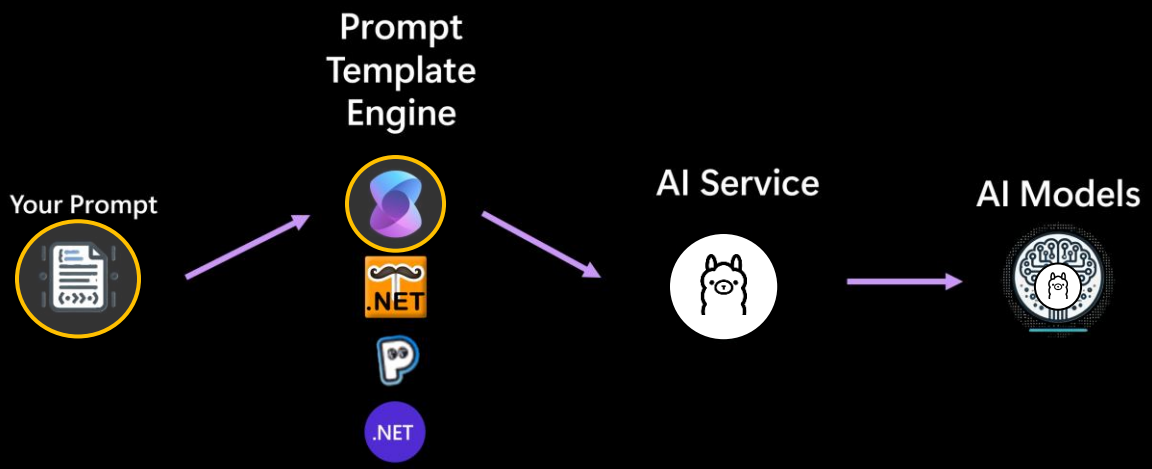
2. Kernel Dependency Injection (Execution)

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.SemanticKernel;

#pragma warning disable SKEXP0070 // Type is for evaluation purposes on

Console.WriteLine("=== Kernel Dependency Injection ===\n");

var modelId = "llama3.2";
var endpoint = new Uri("http://localhost:11434");

var services = new ServiceCollection();
services.AddOllamaChatCompletion(modelId, endpoint);
services.AddKernel();

var serviceProvider = services.BuildServiceProvider();
var kernel = serviceProvider.GetRequiredService<Kernel>();

var prompt = "Why is Neptune blue?";
Console.WriteLine($"User > {prompt}");
Console.Write("Assistant > ");
await foreach (var token in kernel.InvokePromptStreamingAsync(prompt))
{
    Console.Write(token);
};
```



3. Kernel Prompt Template (Basic)

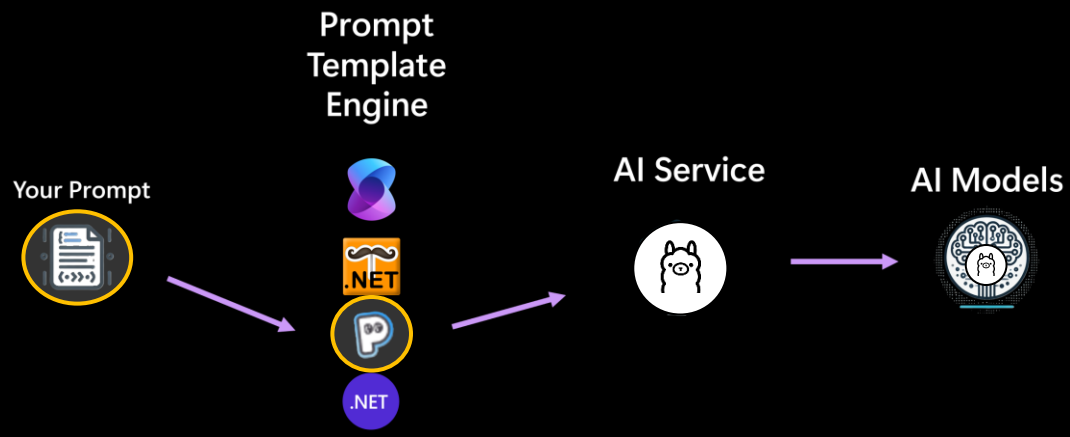
```
1   using Microsoft.SemanticKernel;
2
3   Console.WriteLine("== Kernel Basic Template & Argument variables ===\n");
4
5   var modelId = "phi4";
6   var endpoint = new Uri("http://localhost:11434");
7
8   var kernel = Kernel.CreateBuilder()
9       .AddOllamaChatCompletion(modelId, endpoint)
10      .Build();
11
12  var prompt = """
13      Hello, today is {{$date}}, I'm {{$name}}, and I have {{$age}}.
14      What you thin about my name and what technological advancement happened the year I was born?
15      """;
16
17  var arguments = new KernelArguments()
18  {
19      ["date"] = DateTime.Now.ToString("yyyy-MM-dd"),
20      ["name"] = "Roger",
21      ["age"] = 42
22  };
23
24  Console.WriteLine($"User Template\n---\n{prompt}\n---\n");
25  Console.Write("Assistant > ");
26  await foreach (var token in kernel.InvokePromptStreamingAsync(prompt, arguments))
27  {
28      Console.Write(token);
29  }
```



3. Kernel Prompt Template (Prompty)

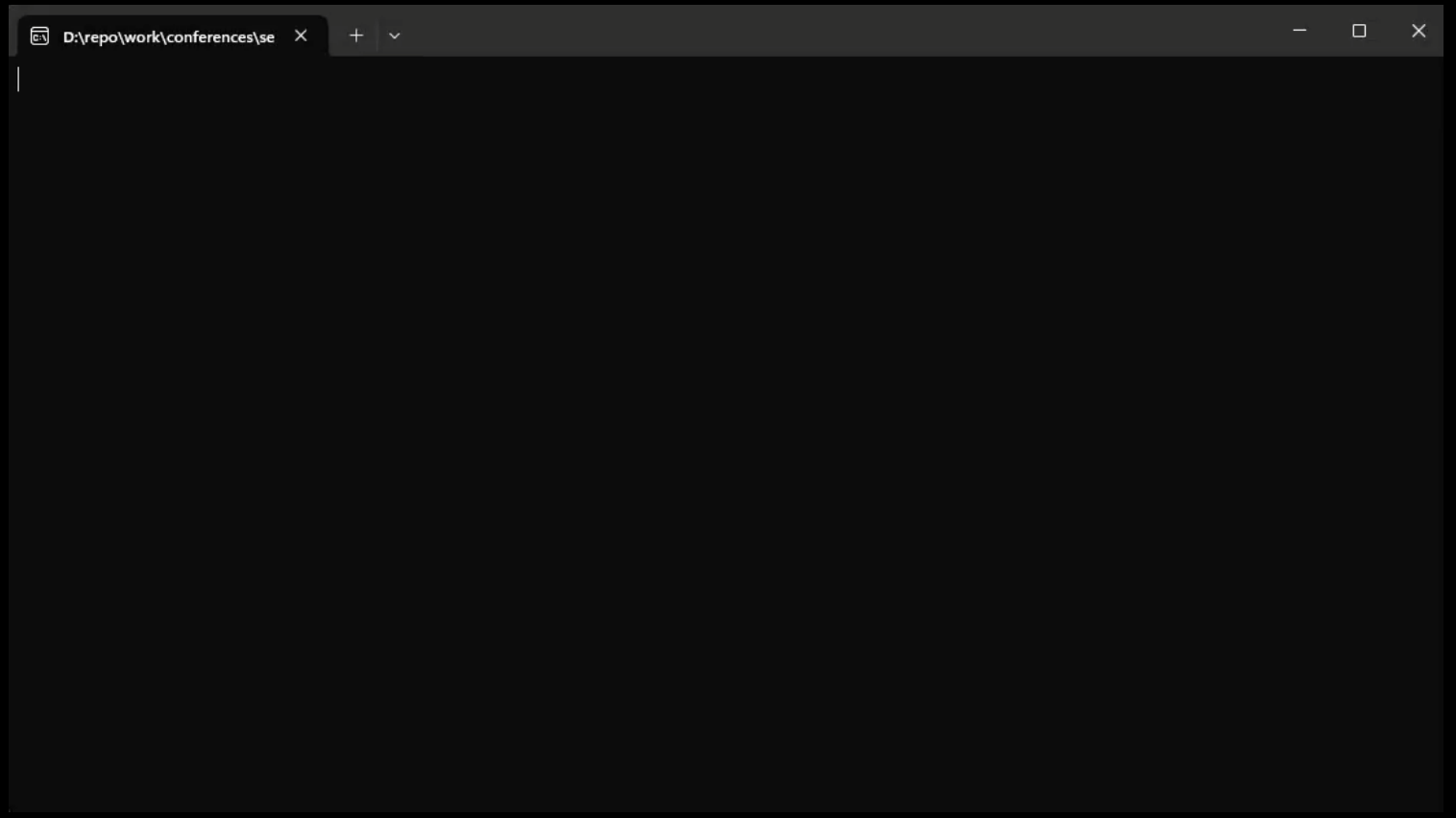
```
17  var arguments = new KernelArguments()
18  {
19      ["date"] = DateTime.Now.ToString("yyyy-MM-dd"),
20      ["name"] = "Roger",
21      ["age"] = 42
22  };
```

```
35  Console.WriteLine("=== Kernel Prompty Template & Argument variables ===\n");
36
37  var promptyTemplate = """
38      ---
39      name: PromptySample
40      ---
41      Hello, today is {{date}}, I'm {{name}}, and I have {{age}}.
42      What you thin about my name and what technological advancement happened the year I was born?
43      """;
44
45  #pragma warning disable SKEXP0040 // Type is for evaluation purposes only and is subject to change
46  var promptyFunction = kernel.CreateFunctionFromPrompty(promptyTemplate);
47
48  Console.WriteLine($"User Template\n{promptyTemplate}\n---\n");
49  Console.Write("Assistant > ");
50  await foreach (var token in kernel.InvokeStreamingAsync(promptyFunction, arguments))
51  {
52      Console.Write(token);
53  }
```



3. Kernel Prompt Template (Basic Exec.)

```
using Microsoft.SemanticKernel;

Console.WriteLine("=== Kernel Basic Template & Argument variables ===\n");

var modelId = "phi4";
var endpoint = new Uri("http://localhost:11434");

var kernel = Kernel.CreateBuilder()
    .AddOllamaChatCompletion(modelId, endpoint)
    .Build();

var prompt = """
    Hello, today is {{$date}}, I'm {{$name}}, and I have {{$age}}.
    What you thin about my name and what technological advancement happened the year I was born?
    """;

var arguments = new KernelArguments()
{
    ["date"] = DateTime.Now.ToString("yyyy-MM-dd"),
    ["name"] = "Roger",
    ["age"] = 42
};

Console.WriteLine($"User Template\n---\n{prompt}\n---\n");
Console.Write("Assistant > ");
await foreach (var token in kernel.InvokePromptStreamingAsync(prompt, arguments))
{
    Console.Write(token);
}
```



3. Kernel Prompt Template (Prompty Exec.)

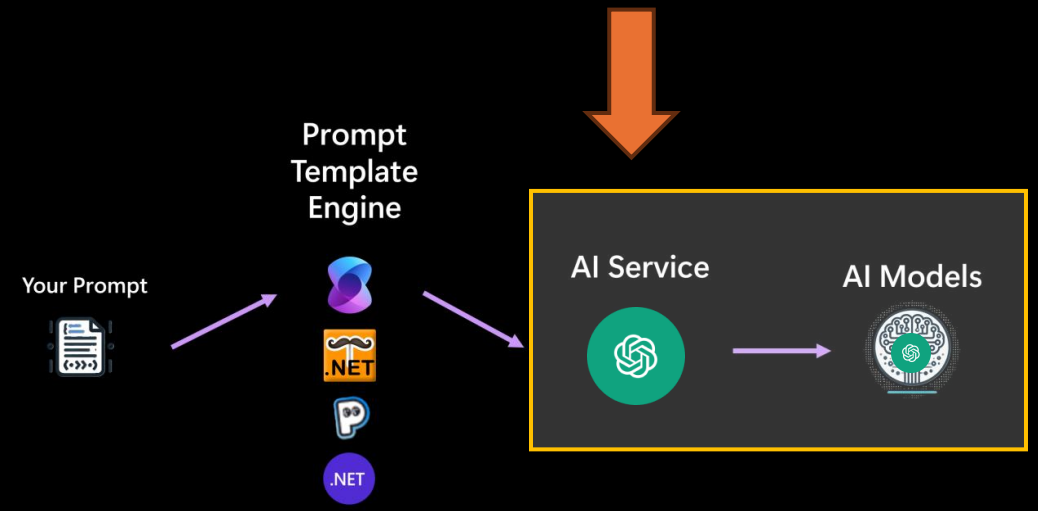
```
17  var arguments = new KernelArguments()
18  {
19      ["date"] = DateTime.Now.ToString("yyyy-MM-dd"),
20      ["name"] = "Roger",
21      ["age"] = 42
22  };
```

```
35  Console.WriteLine("=== Kernel Prompty Template & Argument variables ===\n");
36
37  var promptyTemplate = """
38      ---
39      name: PromptySample
40      ---
41      Hello, today is {{date}}, I'm {{name}}, and I have {{age}}.
42      What you thin about my name and what technological advancement happened the year I was born?
43      """;
44
45  #pragma warning disable SKEXP0040 // Type is for evaluation purposes only and is subject to change
46  var promptyFunction = kernel.CreateFunctionFromPrompty(promptyTemplate);
47
48  Console.WriteLine($"User Template\n{promptyTemplate}\n---\n");
49  Console.Write("Assistant > ");
50  await foreach (var token in kernel.InvokeStreamingAsync(promptyFunction, arguments))
51  {
52      Console.Write(token);
53  }
```

```
D:\repo\work\conferences\se

=== Kernel Prompty Template & Argument variables ===

User Template
---
name: PromptySample
---
Hello, today is {{date}}, I'm {{name}}, and I have {{age}}.
What you thin about my name and what technological advancement happened the year I was born?
---

Assistant > Hi Roger!
```

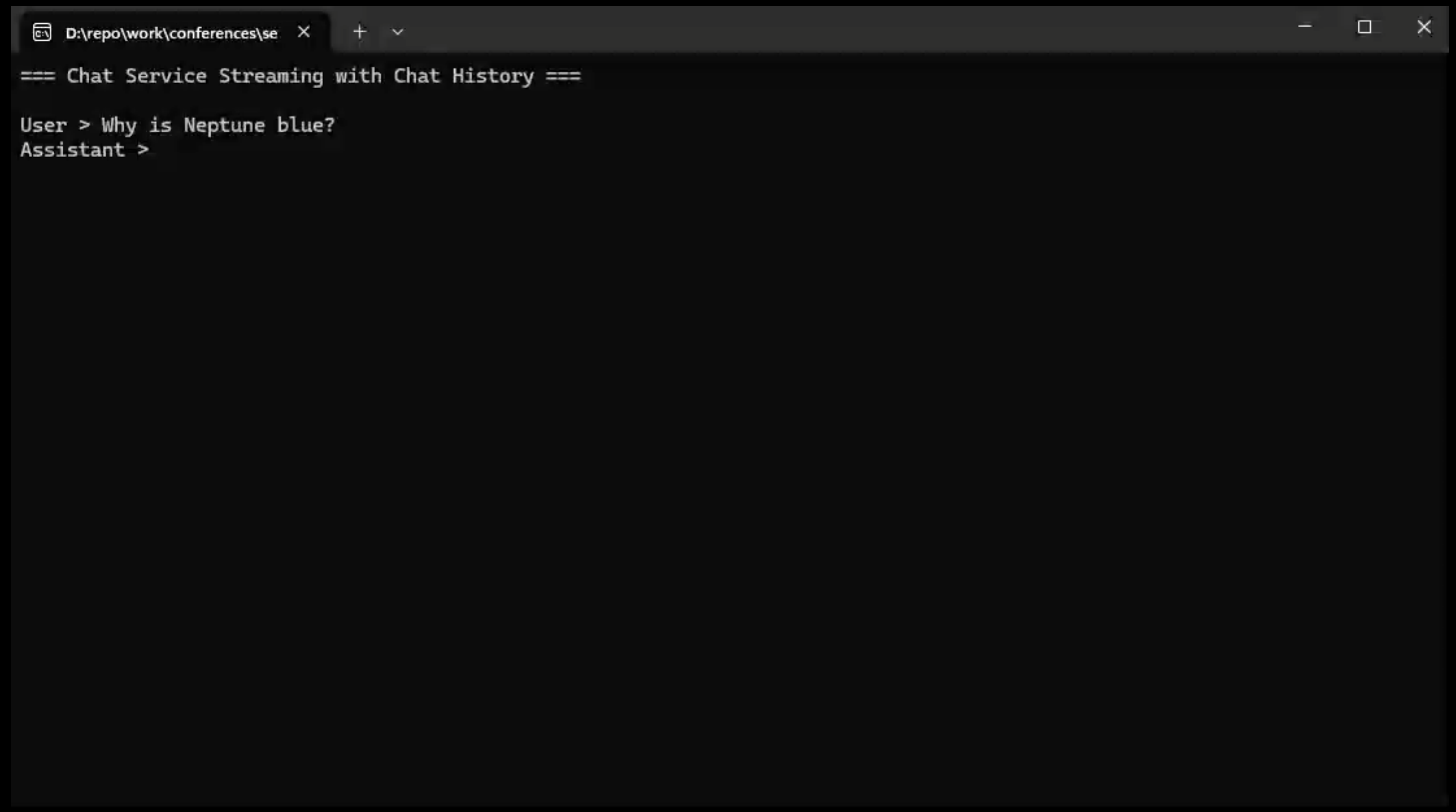
4. Chat Service with Chat History

```
using Microsoft.Extensions.Configuration;
using Microsoft.SemanticKernel.ChatCompletion;
using Microsoft.SemanticKernel.Connectors.OpenAI;

Console.WriteLine("=== Chat Service Streaming with Chat History ===\n");

var apiKey = new ConfigurationBuilder().AddUserSecrets<Program>().Build()["OpenAI:ApiKey"]!;
var modelId = "o1-mini";
var chatService = new OpenAIChatCompletionService(modelId, apiKey);

var userPrompt = "Why is Neptune blue?";

ChatHistory chatHistory = [];
chatHistory.AddSystemMessage("You are presenting in a conference");
chatHistory.AddUserMessage(userPrompt);

Console.WriteLine($"User > {userPrompt}");
Console.Write("Assistant > ");
await foreach (var token in chatService.GetStreamingChatMessageContentsAsync(chatHistory))
{
    Console.Write(token);
}
```



4. Chat Service with Chat History (Execution)

```
1  using Microsoft.Extensions.Configuration;
2  using Microsoft.SemanticKernel.ChatCompletion;
3  using Microsoft.SemanticKernel.Connectors.OpenAI;
4
5  Console.WriteLine("=== Chat Service Streaming with Chat History ===\n");
6
7  var apiKey = new ConfigurationBuilder().AddUserSecrets<Program>().Build()["OpenAI:ApiKey"]!;
8  var modelId = "o1-mini";
9  var chatService = new OpenAIChatCompletionService(modelId, apiKey);
10
11 var userPrompt = "Why is Neptune blue?";
12
13 ChatHistory chatHistory = [];
14 chatHistory.AddSystemMessage("You are presenting in a conference");
15 chatHistory.AddUserMessage(userPrompt);
16
17 Console.WriteLine($"User > {userPrompt}");
18 Console.Write("Assistant > ");
19 await foreach (var token in chatService.GetStreamingChatMessageContentsAsync(chatHistory))
20 {
21     Console.Write(token);
22 }
```

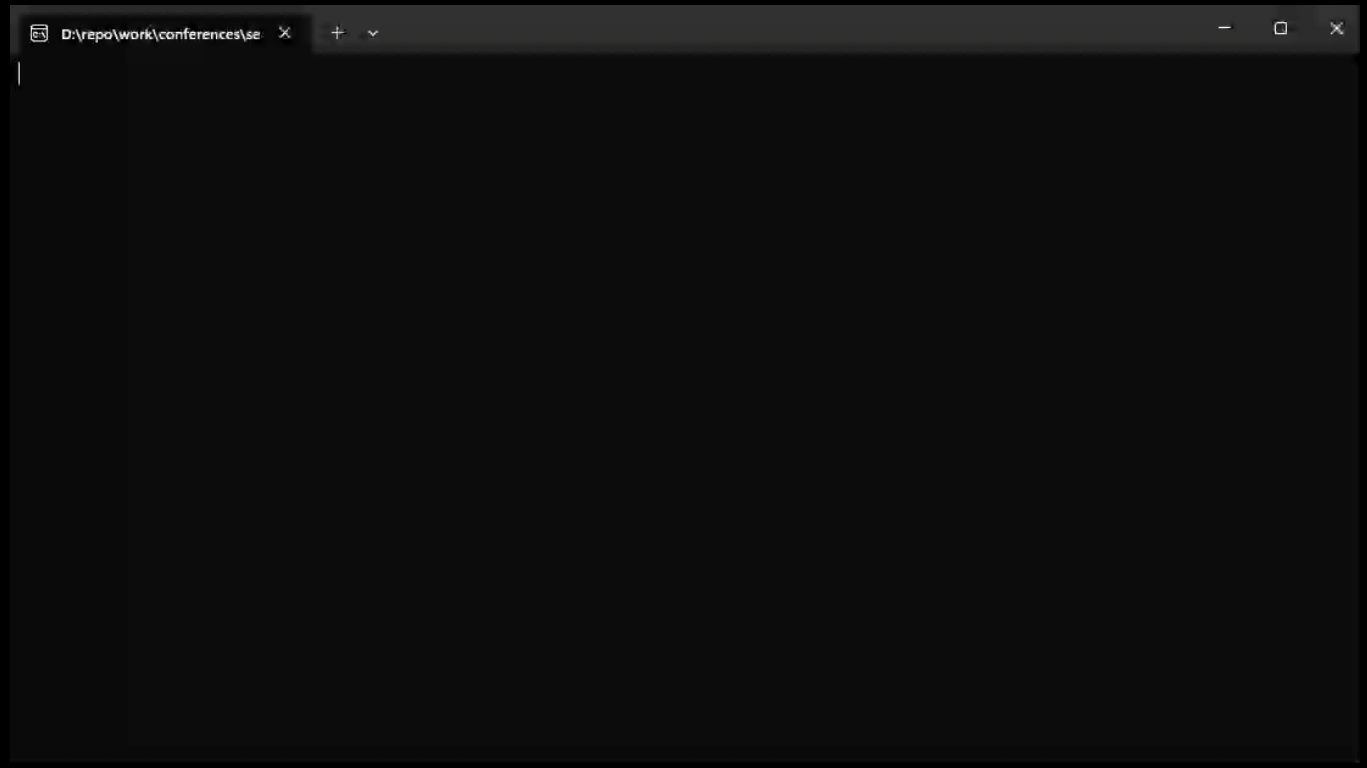


5. Prompt Execution Settings

```
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.ChatCompletion;
using Microsoft.SemanticKernel.Connectors.Ollama;

Console.WriteLine("=== Using prompt execution settings ===\n");

var modelId = "llama3.2";
var endpoint = new Uri("http://localhost:11434");
var prompt = "Write a 5-word sentence describing a sunny day.";

// At a low temperature, you'd expect a straightforward, predictable response like "Warm sunshine fills the air."
var lowTempSettings = new OllamaPromptExecutionSettings { Temperature = 0 };

// At a high temperature, you might get something more unusual or poetic, like "Warmth filled the bright sky."
var highTempSettings = new OllamaPromptExecutionSettings { Temperature = 1.0f };

var kernel = Kernel.CreateBuilder()
    .AddOllamaChatCompletion(modelId, endpoint)
    .Build();

// Get the service from the kernel
var service = kernel.GetRequiredService<IChatCompletionService>();

Console.WriteLine("\nKernel (low temperature):");
await foreach (var token in kernel.InvokePromptStreamingAsync(prompt, new(lowTempSettings)))
{
    Console.Write(token);
}

Console.WriteLine("\n\nChat Service (high temperature):");
await foreach (var token in service.GetStreamingChatMessageContentsAsync(prompt, highTempSettings))
{
    Console.Write(token);
}
```

5. Prompt Execution Settings

```cs
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.ChatCompletion;
using Microsoft.SemanticKernel.Connectors.Ollama;

Console.WriteLine("=== Using prompt execution settings ===\n");

var modelId = "llama3.2";
var endpoint = new Uri("http://localhost:11434");
var prompt = "Write a 5-word sentence describing a sunny day.";

// At a low temperature, you'd expect a straightforward, predictable response like "Warm sunshine fills the air."
var lowTempSettings = new OllamaPromptExecutionSettings { Temperature = 0 };

// At a high temperature, you might get something more unusual or poetic, like "Warmth filled the bright sky."
var highTempSettings = new OllamaPromptExecutionSettings { Temperature = 1.0f };

var kernel = Kernel.CreateBuilder()
    .AddOllamaChatCompletion(modelId, endpoint)
    .Build();

// Get the service from the kernel
var service = kernel.GetRequiredService<IChatCompletionService>();

Console.WriteLine("\nKernel (low temperature):");
await foreach (var token in kernel.InvokePromptStreamingAsync(prompt, new(lowTempSettings)))
{
    Console.Write(token);
}

Console.WriteLine("\n\nChat Service (high temperature):");
await foreach (var token in service.GetStreamingChatMessageContentsAsync(prompt, highTempSettings))
{
    Console.Write(token);
}
```



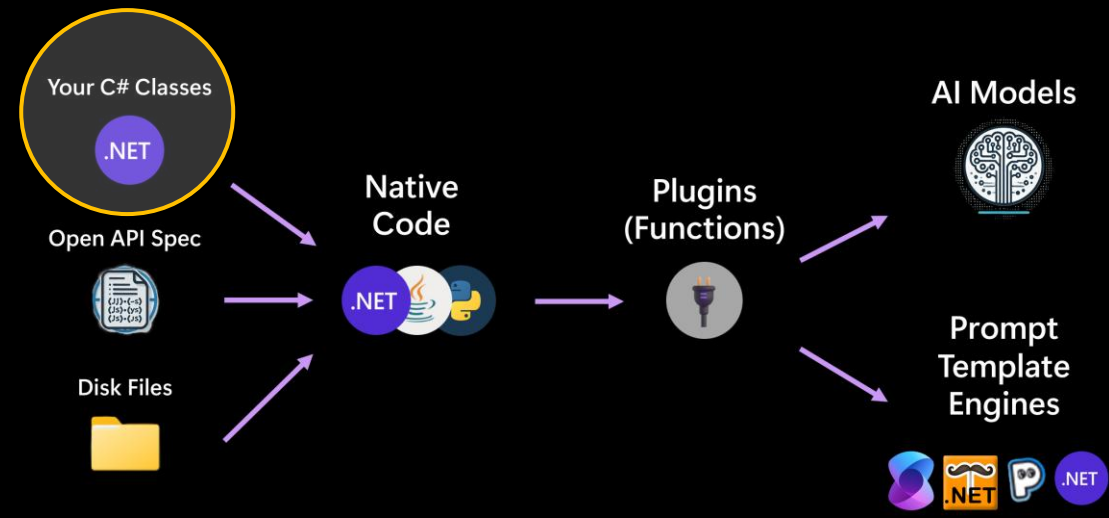
6. Plugins – Template Grounding

Light Bulb Plugin (Stateful)

```
/// <summary>
/// Stateful plugin (It keeps the current bulb state)
/// </summary>
/// <param name="isOn">At the initialization, defines if the bulb is on or off</param>
3 references | Roger Barreto, 23 hours ago | 1 author, 1 change
public class LightBulbPlugin(bool isOn = false)
{
    const string foregroundYellow = "\u001b[93m";
    const string foregroundReset = "\u001b[0m";

    private bool _isOn = isOn;

    [KernelFunction, Description("Checks if the light bulb status is on or off")]
    0 references | Roger Barreto, 23 hours ago | 1 author, 1 change
    public string GetStatus()
    {
        Console.WriteLine($"{foregroundYellow}Plugin Triggered: {nameof(LightBulbPlugin

        return this._isOn ? "on" : "off";
    }

    [KernelFunction, Description("Switches the light bulb status")]
    0 references | Roger Barreto, 23 hours ago | 1 author, 1 change
    public string Switch()
    {
        Console.WriteLine($"{foregroundYellow}Plugin Triggered: {nameof(LightBulbPlugin

        this._isOn = !this._isOn;
        return "switches";
```

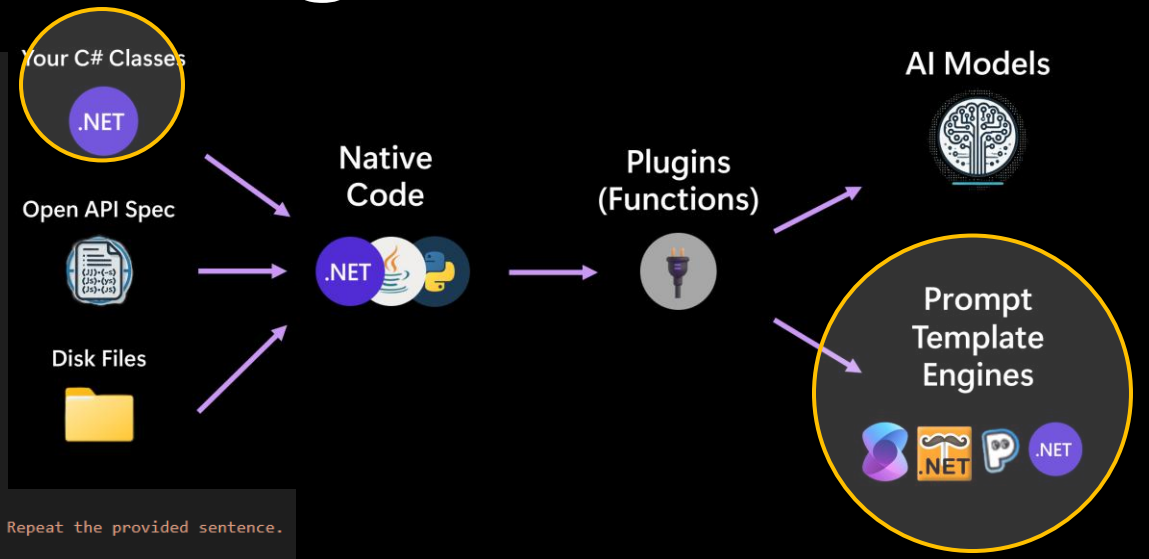
Time Plugin (Stateless)

```
/// <summary>
/// Stateless plugin (It does not keep state)
/// </summary>
2 references | Roger Barreto, 23 hours ago | 1 author, 1 change
public class TimePlugin
{
    const string foregroundYellow = "\u001b[93m";
    const string foregroundReset = "\u001b[0m";

    [KernelFunction]
    0 references | Roger Barreto, 23 hours ago | 1 author, 1 change
    public static string GetDateTime()
    {
        Console.WriteLine($"{foregroundYellow}Plugin Trigge

        var currentTime = DateTime.UtcNow.ToString("R");
        return currentTime;
    }
}
```



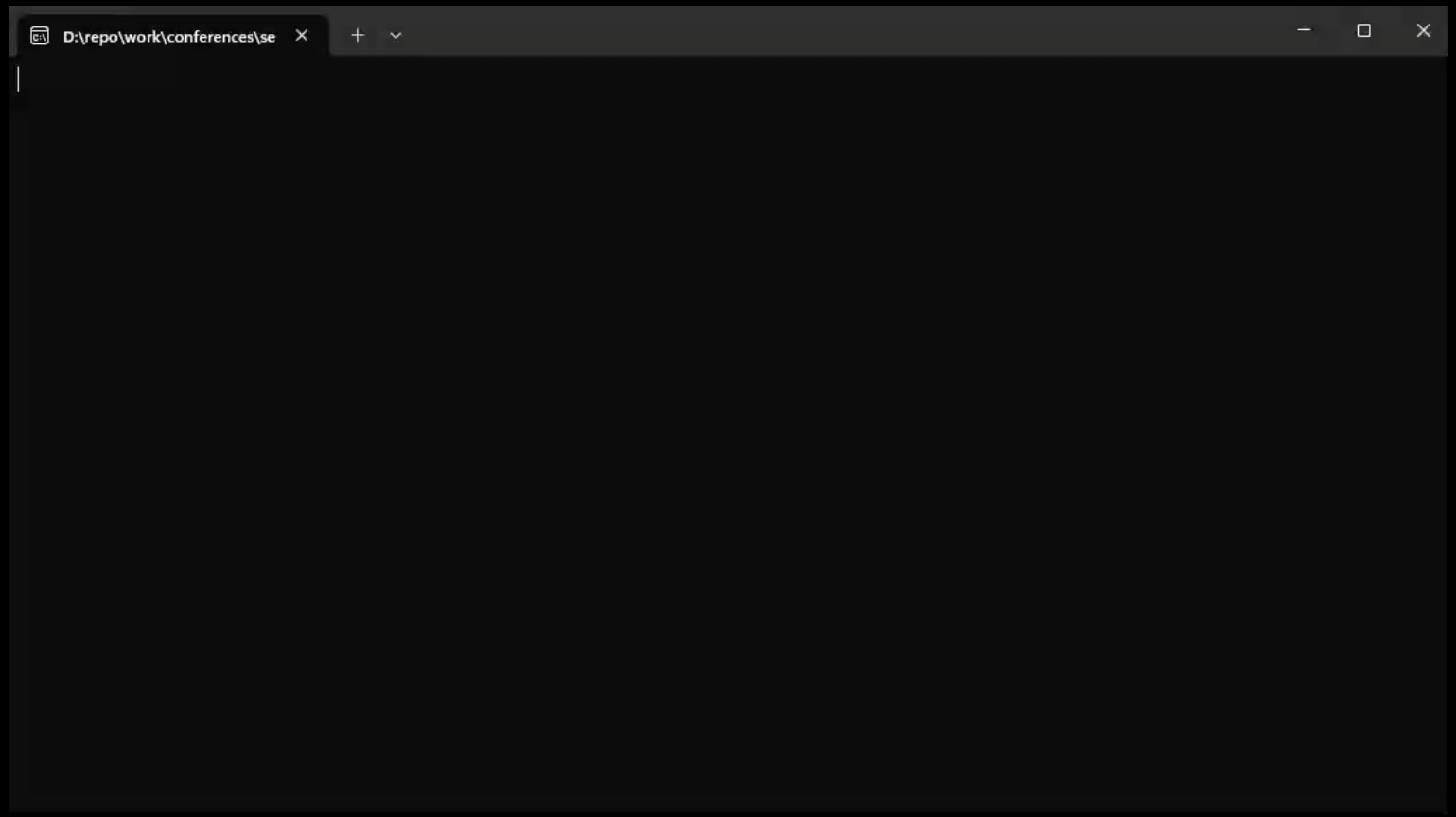
6. Plugins – Template Grounding (Functions)

```csharp
using Microsoft.SemanticKernel;
using Sample;

Console.WriteLine("=== Grounding Prompts with Stateful and Stateless Plugins ===\n");

var modelId = "llama3.2";
var endpoint = new Uri("http://localhost:11434");
var kernel = Kernel.CreateBuilder()
    .AddOllamaChatCompletion(modelId, endpoint)
    .Build();

var myLightBulbPlugin = new LightBulbPlugin(isOn: true);
kernel.Plugins.AddFromObject(myLightBulbPlugin);
kernel.Plugins.AddFromType<TimePlugin>();

var arguments = new KernelArguments
{
    ["name"] = "Roger"
};

var prompt = """
    "At {{GetDateTime}}, {{$name}} looked at the light bulb and see that the light is {{GetStatus}} and then {{Switch}} it.", Repeat the provided sentence.
    """;
Console.WriteLine("Take 1: ");
await foreach (var token in kernel.InvokePromptStreamingAsync(prompt, arguments))
{
    Console.Write(token);
}

Console.WriteLine("\n----");

prompt = """
    "At {{GetDateTime}}, {{$name}} looked at the light bulb and see that the light is {{GetStatus}}.", Repeat the provided sentence.
    """;
Console.WriteLine("Take 2: ");
await foreach (var token in kernel.InvokePromptStreamingAsync(prompt, arguments))
{
    Console.Write(token);
}
```



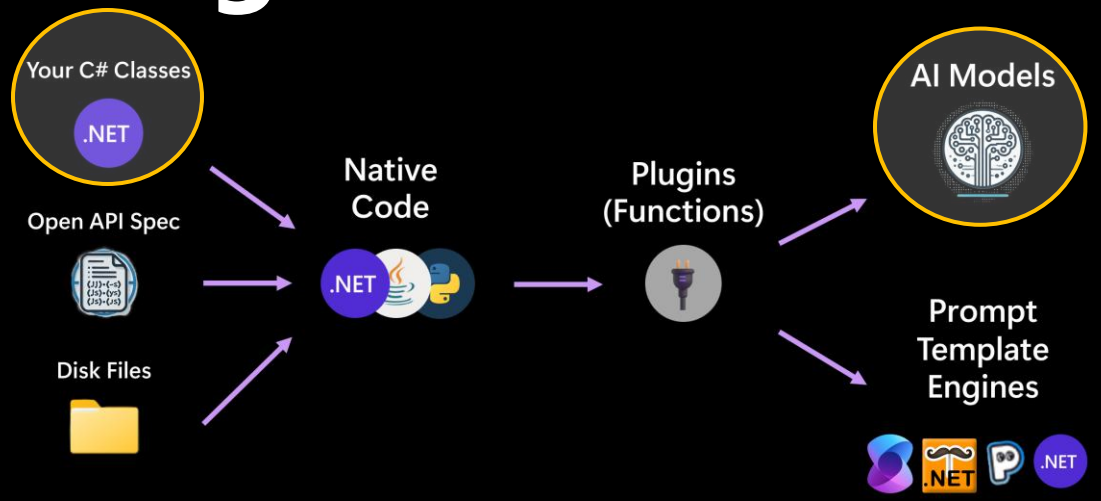
6. Plugins – Template Grounding (Execution)

```
using Microsoft.SemanticKernel;
using Sample;

Console.WriteLine("=== Grounding Prompts with Stateful and Stateless Plugins ===\n");

var modelId = "llama3.2";
var endpoint = new Uri("http://localhost:11434");
var kernel = Kernel.CreateBuilder()
    .AddOllamaChatCompletion(modelId, endpoint)
    .Build();

var myLightBulbPlugin = new LightBulbPlugin(isOn: true);
kernel.Plugins.AddFromObject(myLightBulbPlugin);
kernel.Plugins.AddFromType<TimePlugin>();

var arguments = new KernelArguments
{
    ["name"] = "Roger"
};

var prompt = """
    "At {{GetDateTime}}: {{$name}} looked at the light bulb and see that the light is {{GetStatus}} and then {{Switch}} it.", Repeat the provided sentence.
    """;
Console.WriteLine("Take 1: ");
await foreach (var token in kernel.InvokePromptStreamingAsync(prompt, arguments))
{
    Console.Write(token);
}

Console.WriteLine("\n----");

prompt = """
    "At {{GetDateTime}}: {{$name}} looked at the light bulb and see that the light is {{GetStatus}}.", Repeat the provided sentence.
    """;
Console.WriteLine("Take 2: ");
await foreach (var token in kernel.InvokePromptStreamingAsync(prompt, arguments))
{
    Console.Write(token);
}
```



7. Plugins – AI Function Calling

```cs
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.ChatCompletion;
using Microsoft.SemanticKernel.Connectors.Ollama;
using Sample;

Console.WriteLine("=== Plugin Function Calling with Stateful and Stateless Plugins ===");

var modelId = "llama3.2";
var endpoint = new Uri("http://localhost:11434");
var kernel = Kernel.CreateBuilder()
    .AddOllamaChatCompletion(modelId, endpoint)
    .Build();

var lightBulbPlugin = new LightBulbPlugin(isOn: true);
kernel.Plugins.AddFromObject(lightBulbPlugin);
kernel.Plugins.AddFromType<TimePlugin>();

var service = kernel.GetRequiredService<IChatCompletionService>();

var settings = new OllamaPromptExecutionSettings { FunctionChoiceBehavior = FunctionChoiceBehavior.Auto() };

var prompt = "What is the light status and what time of the day is it?";
Console.WriteLine($"\nUser: {prompt}");

var result = await kernel.InvokePromptAsync(prompt, new(settings));
Console.WriteLine($"Assistant: {result}");

prompt = "Please switch the light.";
Console.WriteLine($"\nUser: {prompt}");

result = await kernel.InvokePromptAsync(prompt, new(settings));
Console.WriteLine($"Assistant: {result}");

prompt = "What is the light bulb status and what time of the day is it?";
Console.WriteLine($"\nUser: {prompt}");

var contentResult = await service.GetChatMessageContentAsync(prompt, settings, kernel);
Console.WriteLine($"Assistant: {contentResult}");
```

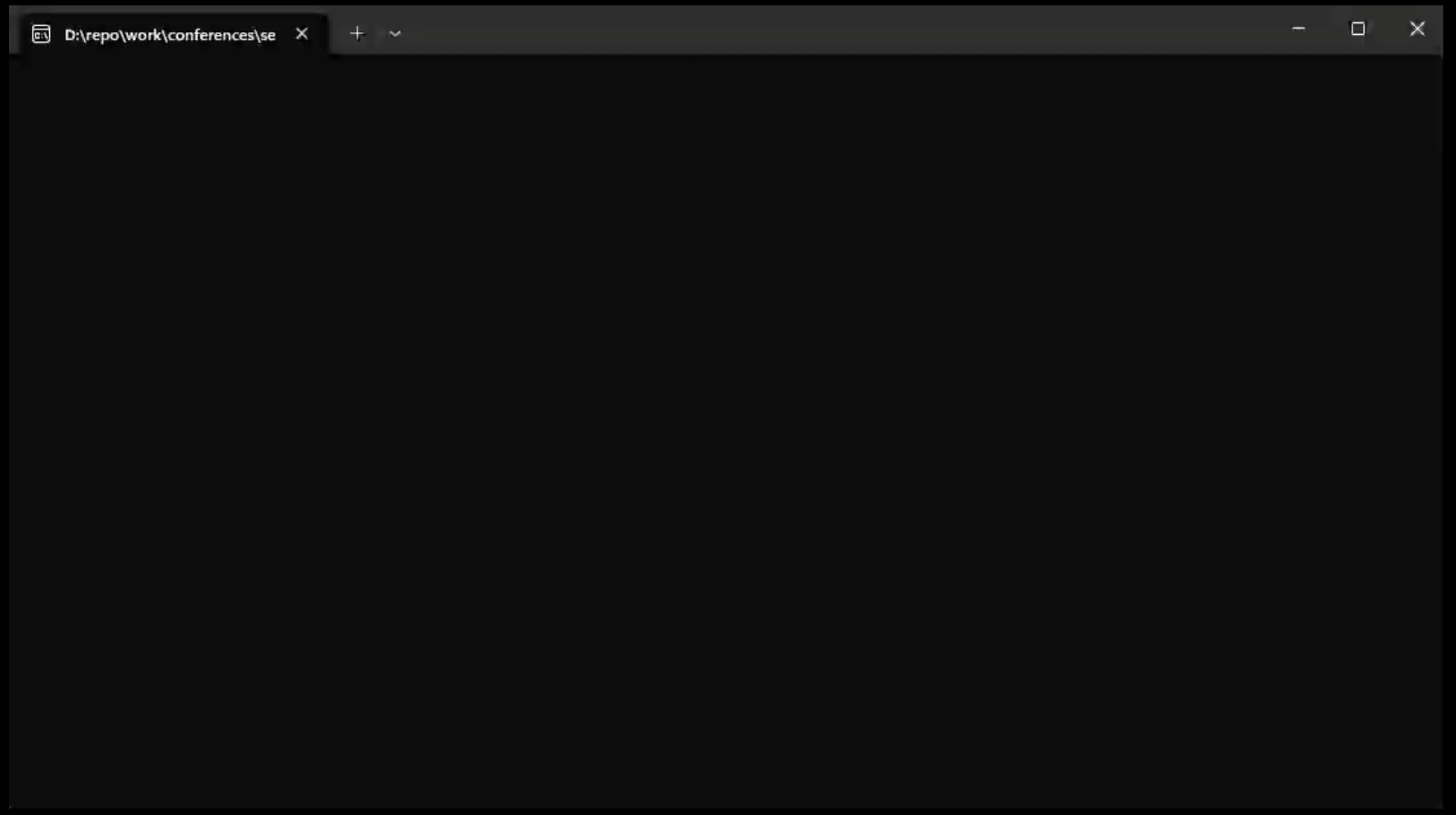


Light Bulb Plugin (Stateful)

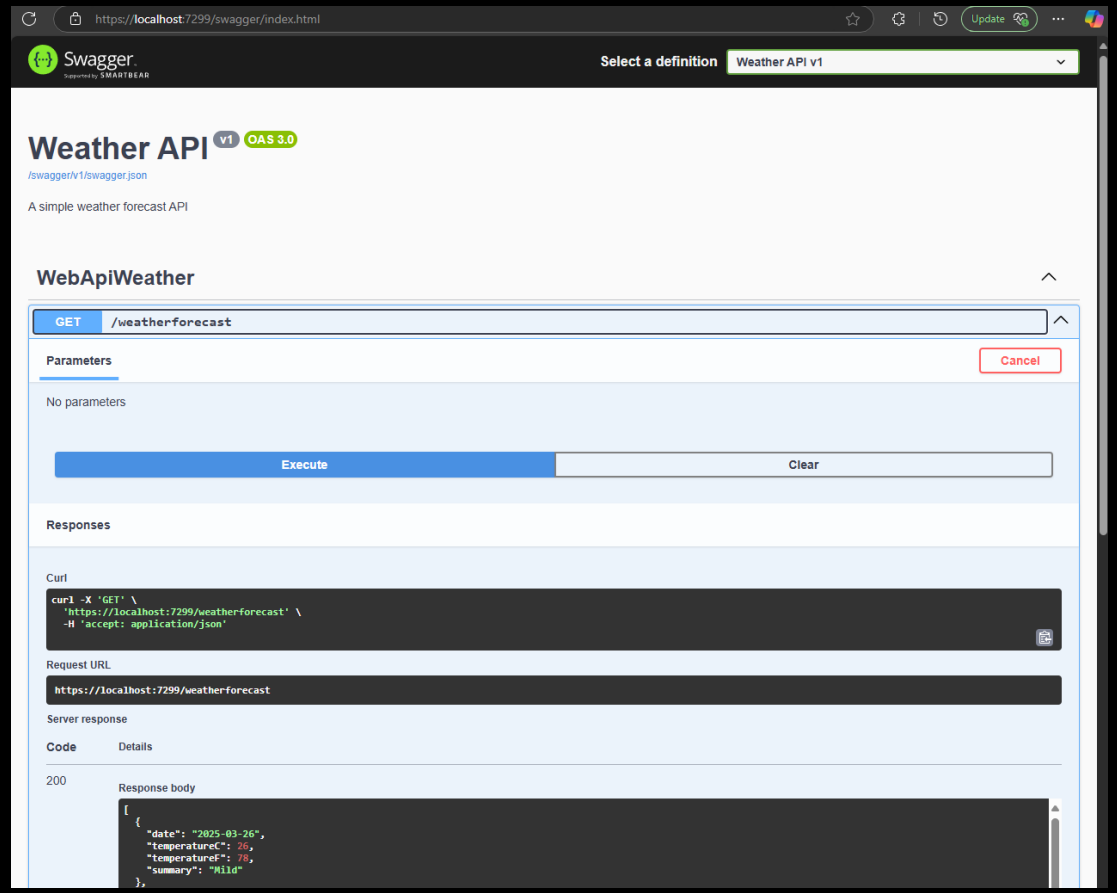
```cs
3 references | Roger Barreto, 1 day ago | 1 author, 1 change
public class LightBulbPlugin(bool isOn = false)
{
    [KernelFunction, Description("Checks if the light bulb status is on or off")]
    0 references | Roger Barreto, 1 day ago | 1 author, 1 change
    public string GetStatus()...

    [KernelFunction, Description("Switches the light bulb status")]
    0 references | Roger Barreto, 1 day ago | 1 author, 1 change
    public string Switch()...
```

7. Plugins – AI Function Calling (Execution)

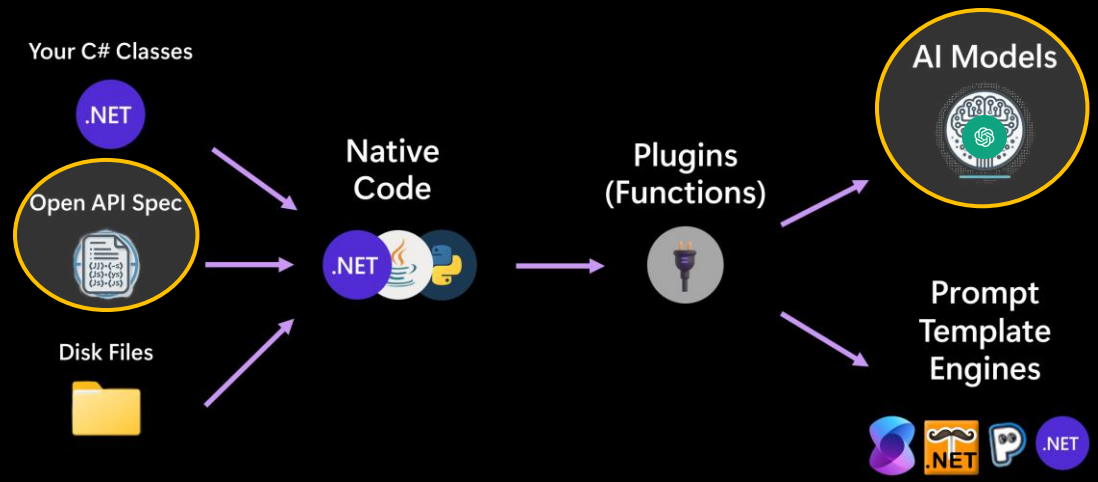
```csharp
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.ChatCompletion;
using Microsoft.SemanticKernel.Connectors.Ollama;
using Sample;

Console.WriteLine("=== Plugin Function Calling with Stateful and Stateless Plugins ===");

var modelId = "llama3.2";
var endpoint = new Uri("http://localhost:11434");
var kernel = Kernel.CreateBuilder()
    .AddOllamaChatCompletion(modelId, endpoint)
    .Build();

var lightBulbPlugin = new LightBulbPlugin(isOn: true);
kernel.Plugins.AddFromObject(lightBulbPlugin);
kernel.Plugins.AddFromType<TimePlugin>();

var service = kernel.GetRequiredService<IChatCompletionService>();

var settings = new OllamaPromptExecutionSettings { FunctionChoiceBehavior = FunctionChoiceBehavior.Auto() };

var prompt = "What is the light status and what time of the day is it?";
Console.WriteLine($"\nUser: {prompt}");

var result = await kernel.InvokePromptAsync(prompt, new(settings));
Console.WriteLine($"Assistant: {result}");

prompt = "Please switch the light.";
Console.WriteLine($"\nUser: {prompt}");

result = await kernel.InvokePromptAsync(prompt, new(settings));
Console.WriteLine($"Assistant: {result}");

prompt = "What is the light bulb status and what time of the day is it?";
Console.WriteLine($"\nUser: {prompt}");

var contentResult = await service.GetChatMessageContentAsync(prompt, settings, kernel);
Console.WriteLine($"Assistant: {contentResult}");
```



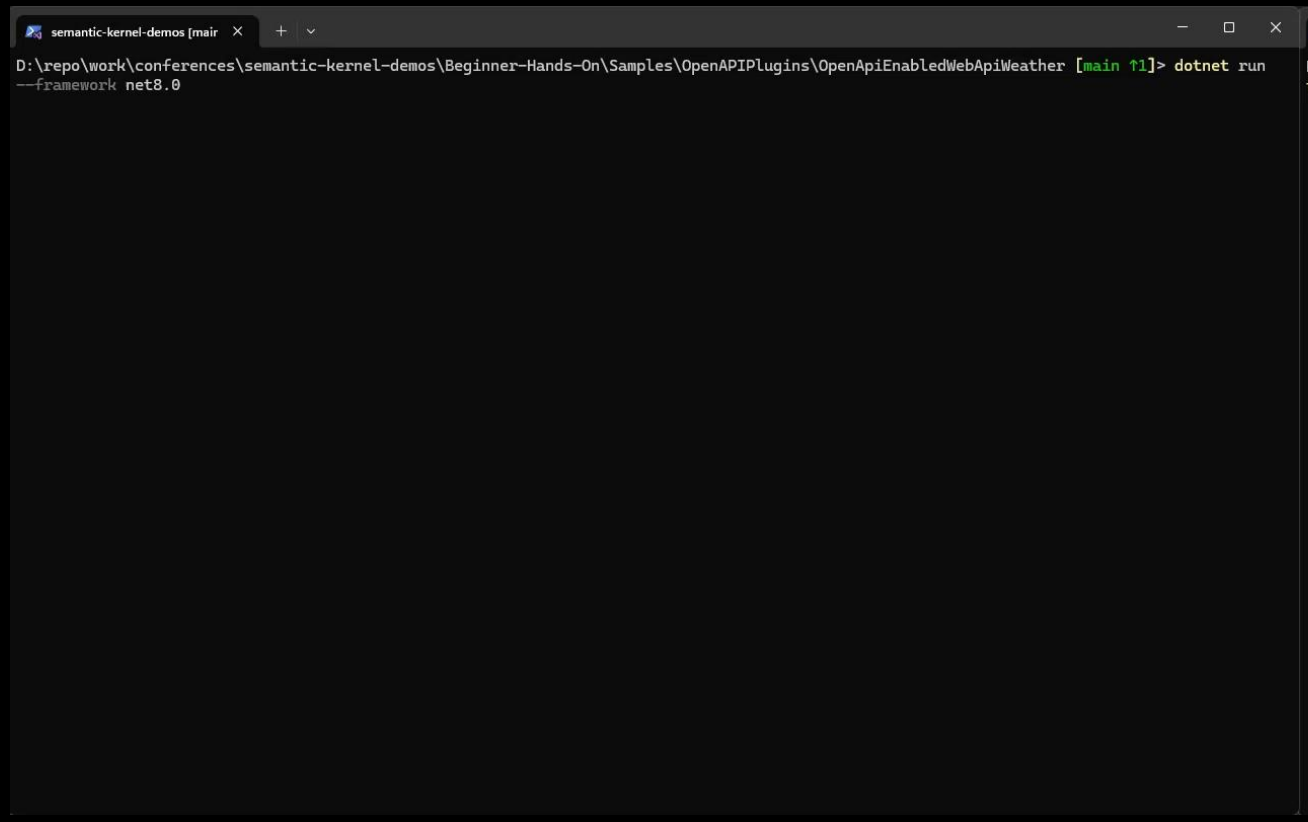
8. OpenApi Plugins – AI Function Calling

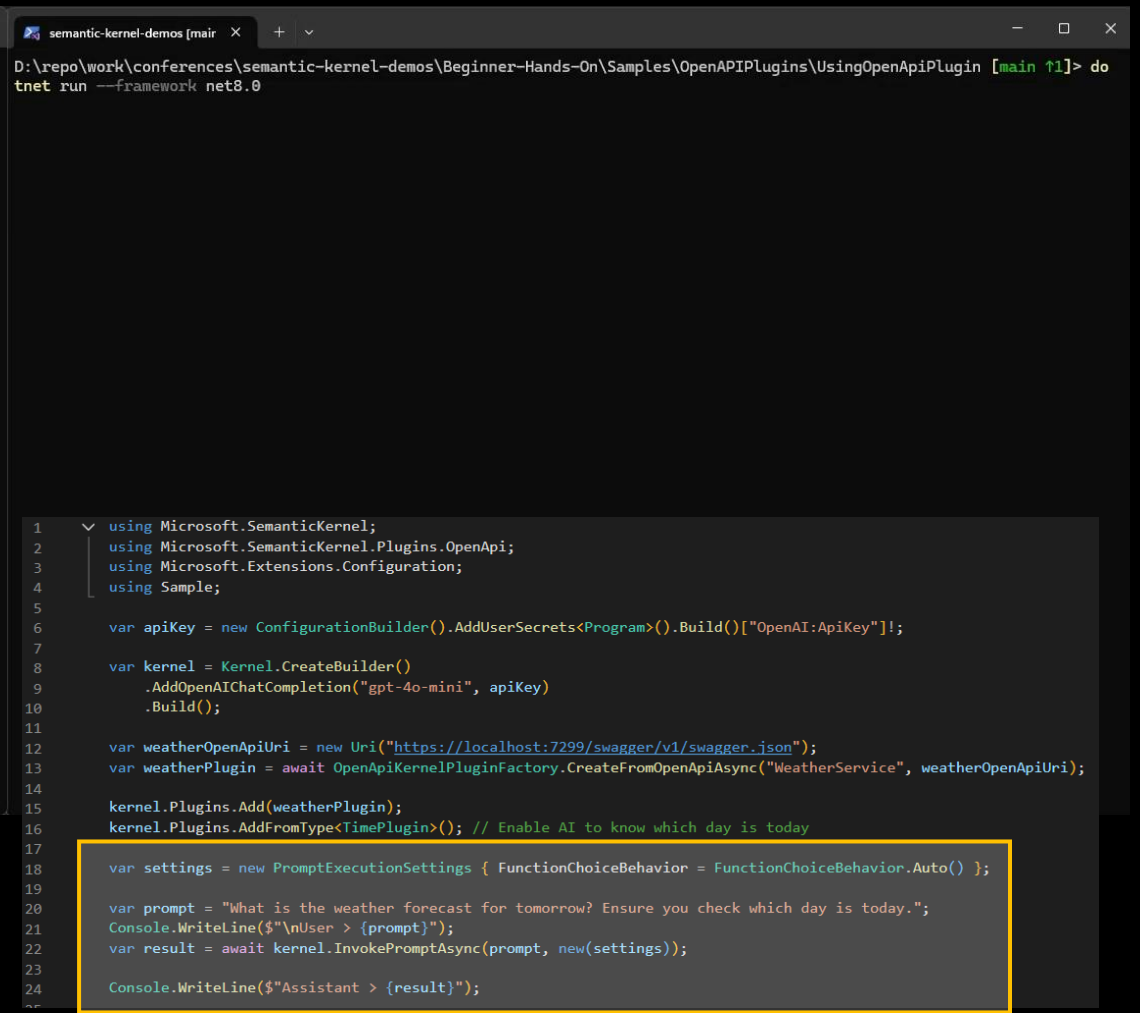



```
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Plugins.OpenApi;
using Microsoft.Extensions.Configuration;
using Sample;

var apiKey = new ConfigurationBuilder().AddUserSecrets<Program>().Build()["OpenAI:ApiKey"]!;

var kernel = Kernel.CreateBuilder()
    .AddOpenAIChatCompletion("gpt-4o-mini", apiKey)
    .Build();

var weatherOpenApiUri = new Uri("https://localhost:7299/swagger/v1/swagger.json");
var weatherPlugin = await OpenApiKernelPluginFactory.CreateFromOpenApiAsync("WeatherService", weatherOpenApiUri);

kernel.Plugins.Add(weatherPlugin);
kernel.Plugins.AddFromType<TimePlugin>(); // Enable AI to know which day is today

var settings = new PromptExecutionSettings { FunctionChoiceBehavior = FunctionChoiceBehavior.Auto() };

var prompt = "What is the weather forecast for tomorrow? Ensure you check which day is today.";
Console.WriteLine($"\nUser > {prompt}");
var result = await kernel.InvokePromptAsync(prompt, new(settings));

Console.WriteLine($"Assistant > {result}");
```



8. OpenApi Plugins – AI Function Calling (Exec)






```
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Plugins.OpenApi;
using Microsoft.Extensions.Configuration;
using Sample;

var apiKey = new ConfigurationBuilder().AddUserSecrets<Program>().Build()["OpenAI:ApiKey"]!;

var kernel = Kernel.CreateBuilder()
    .AddOpenAIChatCompletion("gpt-4o-mini", apiKey)
    .Build();

var weatherOpenApiUri = new Uri("https://localhost:7299/swagger/v1/swagger.json");
var weatherPlugin = await OpenApiKernelPluginFactory.CreateFromOpenApiAsync("WeatherService", weatherOpenApiUri);

kernel.Plugins.Add(weatherPlugin);
kernel.Plugins.AddFromType<TimePlugin>(); // Enable AI to know which day is today

var settings = new PromptExecutionSettings { FunctionChoiceBehavior = FunctionChoiceBehavior.Auto() };

var prompt = "What is the weather forecast for tomorrow? Ensure you check which day is today.";
Console.WriteLine($"\nUser > {prompt}");
var result = await kernel.InvokePromptAsync(prompt, new(settings));

Console.WriteLine($"Assistant > {result}");
```

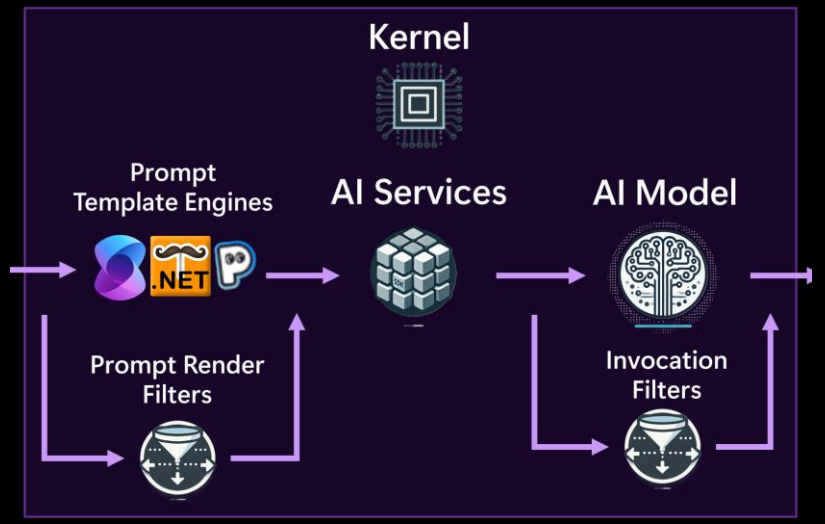
9. Kernel Filters – Render & Function Invocation

Prompt Rendering

```
/// <summary>
/// This filter is used to intercept before and after a kernel prompt is rendered and write to the console its name and result.
/// </summary>
1 reference | Roger Barreto, 1 day ago | 1 author, 1 change
public class ConsoleWritePromptRenderingFilter : IPromptRenderFilter
{
    0 references | Roger Barreto, 1 day ago | 1 author, 1 change
    public async Task OnPromptRenderAsync(PromptRenderContext context, Func<PromptRenderContext, Task> next)
    {
        // Before prompt was rendered
        await next.Invoke(context);

        // After prompt was rendered
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine($"Prompt Rendered: '{context.RenderedPrompt}'");
        Console.ResetColor();

        // Write the prompt to the console as is with no color formatting (Similar to assistant output)
        Console.WriteLine($"{context.RenderedPrompt}");
    }
}
```

Function Invocation

```
/// <summary>
/// This filter is used to intercept before and after a kernel function is invoked and write to the console its name and result.
/// </summary>
1 reference | Roger Barreto, 1 day ago | 1 author, 1 change
public class ConsoleWriteFunctionInvocationFilter : IFunctionInvocationFilter
{
    0 references | Roger Barreto, 1 day ago | 1 author, 1 change
    public async Task OnFunctionInvocationAsync(FunctionInvocationContext context, Func<FunctionInvocationContext, Task> next)
    {
        string writeMessage = $"Function Invoked: '{context.Function.Name}'";
        if (!string.IsNullOrEmpty(context.Function.PluginName))
        {
            writeMessage += $" - Plugin: '{context.Function.PluginName}'";
        }

        Console.ForegroundColor = ConsoleColor.Cyan;
        Console.WriteLine(writeMessage);
        Console.ResetColor();

        await next.Invoke(context);

        // On this sample all functions without a plugin name are AI model responses
        if (string.IsNullOrEmpty(context.Function.PluginName))
        {
            Console.WriteLine($"Assistant: {context.Result}");
        }
    }
```

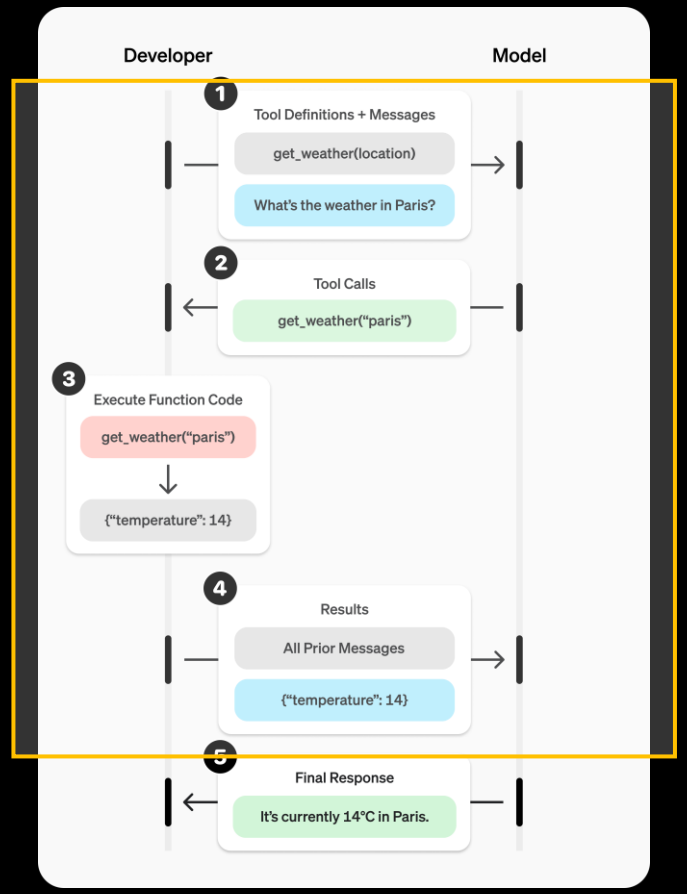


9. Kernel Filters – Auto Function Invocation

Auto Function Invocation (AI Function Calling Loop)

```
6   /// <summary>
7   /// This filter is used specifically for function calling models where your want to intercept
8   // AI Model function calls and trace (Console.Write) or change the behavior.
9   /// </summary>
    1 reference | Roger Barreto, 1 day ago | 1 author, 1 change
10  public class ConsoleWriteAutoFunctionInvocationFilter : IAutoFunctionInvocationFilter
11  {
        0 references | Roger Barreto, 1 day ago | 1 author, 1 change
12      public async Task OnAutoFunctionInvocationAsync(AutoFunctionInvocationContext context, Func<AutoFun
13      {
14          Console.ForegroundColor = ConsoleColor.DarkYellow;
15          Console.WriteLine(string.Concat($"Auto Function Invoked: '{context.Function.Name}'",
16
17                                          // Request sequence index identifies what and how many requests
18                                          $" - Request sequence index: {context.RequestSequenceIndex}",
19
20                                          // Function sequence index identifies what and which is the cur
21                                          // Note: Multiple functions can be called in a single request (
22                                          $" - Function sequence index: {context.FunctionSequenceIndex}",
23
24                                          // Total number of functions requested to be called by the AI M
25                                          $" - Total number of functions: {context.FunctionCount}"));
26          Console.ResetColor();
27
28          await next.Invoke(context);
29      }
30  }
```

AI Function Calling Loop



9. Kernel Filters – Setup & Execution

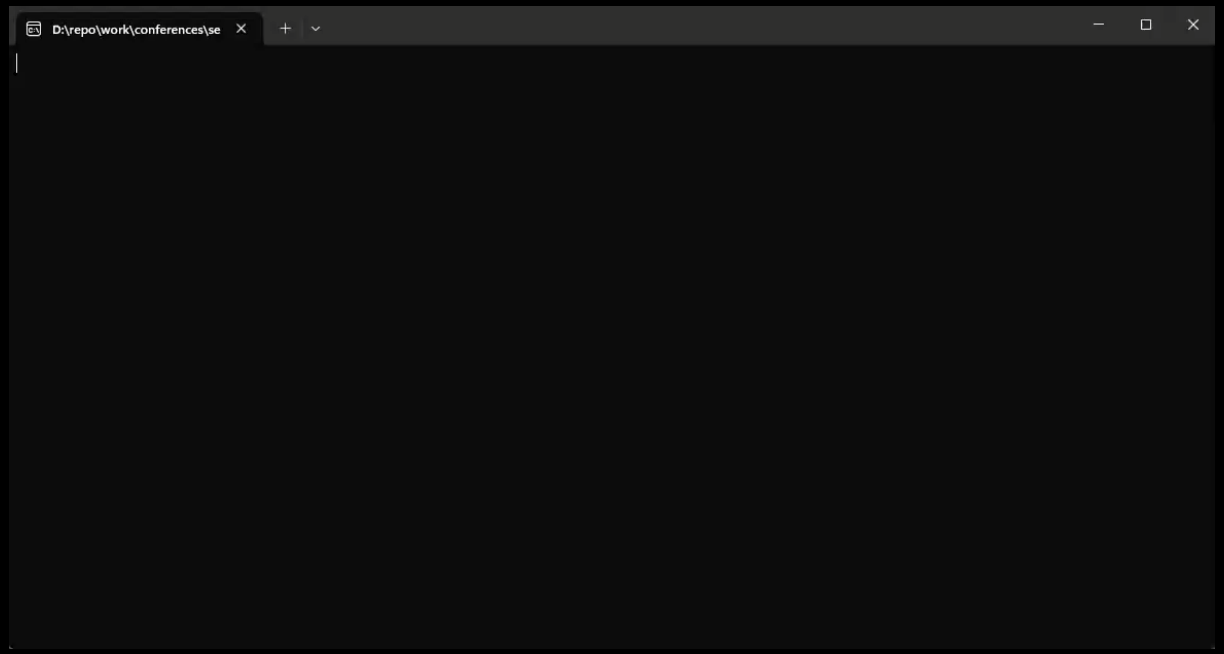
Adding directly to a Kernel instance

```
using Microsoft.Extensions.Configuration;
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.ChatCompletion;
using Microsoft.SemanticKernel.Connectors.OpenAI;
using Sample;

var apiKey = new ConfigurationBuilder().AddUserSecrets<Program>().Build()["OpenAI:ApiKey"]!;

Console.WriteLine("=== Using Kernel Filters ===");

var lightBulbPlugin = new LightBulbPlugin(isOn: true);

var modelId = "gpt-4o-mini";
var kernel = Kernel.CreateBuilder()
    .AddOpenAIChatCompletion(modelId, apiKey)
    .Build();

// Add the custom function invocation filter
kernel.FunctionInvocationFilters.Add(new ConsoleWriteFunctionInvocationFilter());

// Add the custom prompt rendering filter
kernel.PromptRenderFilters.Add(new ConsoleWritePromptRenderingFilter());

// Add the custom auto function invocation filter
kernel.AutoFunctionInvocationFilters.Add(new ConsoleWriteAutoFunctionInvocationFilter());

kernel.Plugins.AddFromObject(lightBulbPlugin);
kernel.Plugins.AddFromType<TimePlugin>();

var service = kernel.GetRequiredService<IChatCompletionService>();

var settings = new OpenAIPromptExecutionSettings { FunctionChoiceBehavior = FunctionChoiceBehavior.Auto() };
var arguments = new KernelArguments(settings) { ["name"] = "Roger" };

var prompt = "{{$name}}: What is the light status and what time of the day is it?";
await kernel.InvokePromptAsync(prompt, arguments);

prompt = "{{$name}}: Please switch the light.";
var promptFunction = kernel.CreateFunctionFromPrompt(prompt, functionName: "FunctionFromPrompt");
await kernel.InvokeAsync(promptFunction, arguments);

// Direct service calls don't trigger prompt rendering
prompt = $"{arguments["name"]}: What is the light bulb status and what time of the day is it?";
Console.WriteLine($"\n{prompt}");
var kernelContent = await service.GetChatMessageContentAsync(prompt, settings, kernel);
Console.WriteLine($"Assistant: {kernelContent}");

```

Adding via Dependency Injection

```
var builder = Kernel.CreateBuilder()
    .AddOpenAIChatCompletion(modelId, apiKey);

builder.Services.AddSingleton<IFunctionInvocationFilter, ConsoleWriteFunctionInvocationFilter>();
builder.Services.AddSingleton<IPromptRenderFilter, ConsoleWritePromptRenderingFilter>();
builder.Services.AddSingleton<IAutoFunctionInvocationFilter, ConsoleWriteAutoFunctionInvocationFilter>();

var kernel = builder.Build();
```

Execution

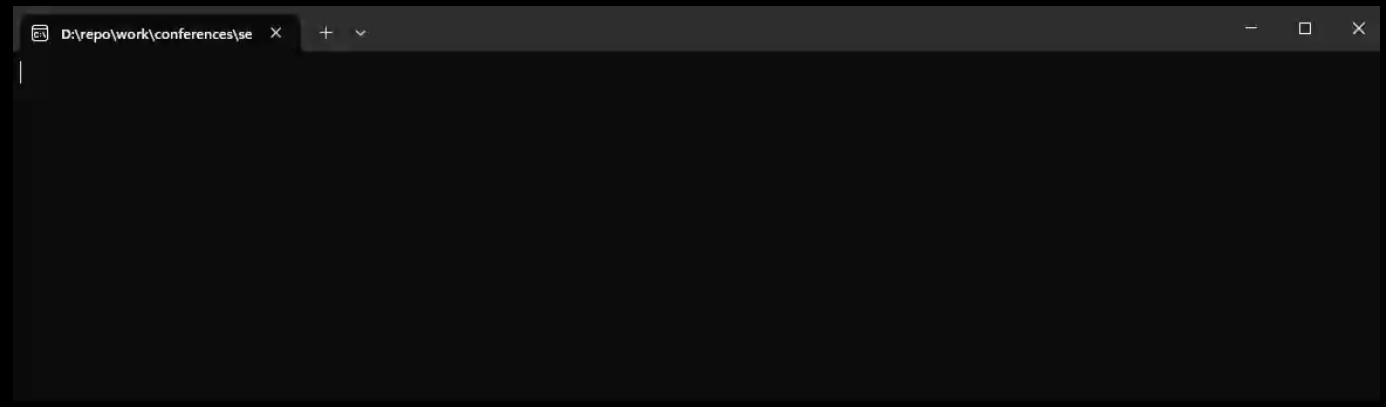


10. Open Telemetry + Aspire Dashboard

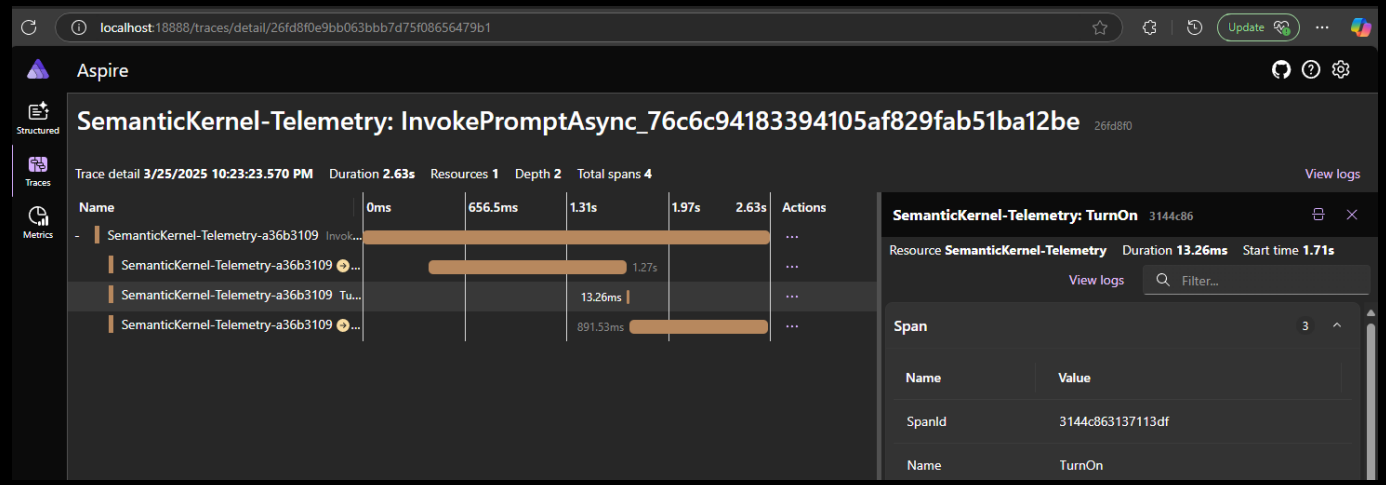
Setup: Resource + Tracer + Meter + Logger Factory

```
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.SemanticKernel;
using OpenTelemetry;
using OpenTelemetry.Logs;
using OpenTelemetry.Metrics;
using OpenTelemetry.Resources;
using OpenTelemetry.Trace;
using Sample;

Console.WriteLine("=== Open Telemetry Aspire Dashboard ===\n\n");

var builder = Kernel.CreateBuilder();
var apiKey = new ConfigurationBuilder().AddUserSecrets<Program>().Build()["OpenAI:ApiKey"]!;
var modelId = "gpt-4o-mini";
var oTelExporterEndpoint = "http://localhost:4317";

var resourceBuilder = ResourceBuilder
    .CreateDefault()
    .AddService("SemanticKernel-Telemetry");

// Enable model diagnostics with sensitive data.
AppContext.SetSwitch("Microsoft.SemanticKernel.Experimental.GenAI.EnableOTelDiagnosticsSensitive", true);

// Configuring the tracer provider
using var traceProvider = Sdk.CreateTracerProviderBuilder()
    .SetResourceBuilder(resourceBuilder)
    .AddSource("Microsoft.SemanticKernel*")
    .AddOtlpExporter(options => options.Endpoint = new Uri(oTelExporterEndpoint))
    .Build();

// Configuring the meter provider
using var meterProvider = Sdk.CreateMeterProviderBuilder()
    .SetResourceBuilder(resourceBuilder)
    .AddMeter("Microsoft.SemanticKernel*")
    .AddOtlpExporter(options => options.Endpoint = new Uri(oTelExporterEndpoint))
    .Build();

// Configuring the logger factory
using var loggerFactory = LoggerFactory.Create(builder =>
{
    // Add OpenTelemetry as a logging provider
    builder.AddOpenTelemetry(options =>
    {
        options.SetResourceBuilder(resourceBuilder);
        options.AddOtlpExporter(options => options.Endpoint = new Uri(oTelExporterEndpoint));
        // Format log messages. This is default to false.
        options.IncludeFormattedMessage = true;
        options.IncludeScopes = true;
    });
    builder.SetMinimumLevel(LogLevel.Information);
});
```

Execution



Aspire Dashboard



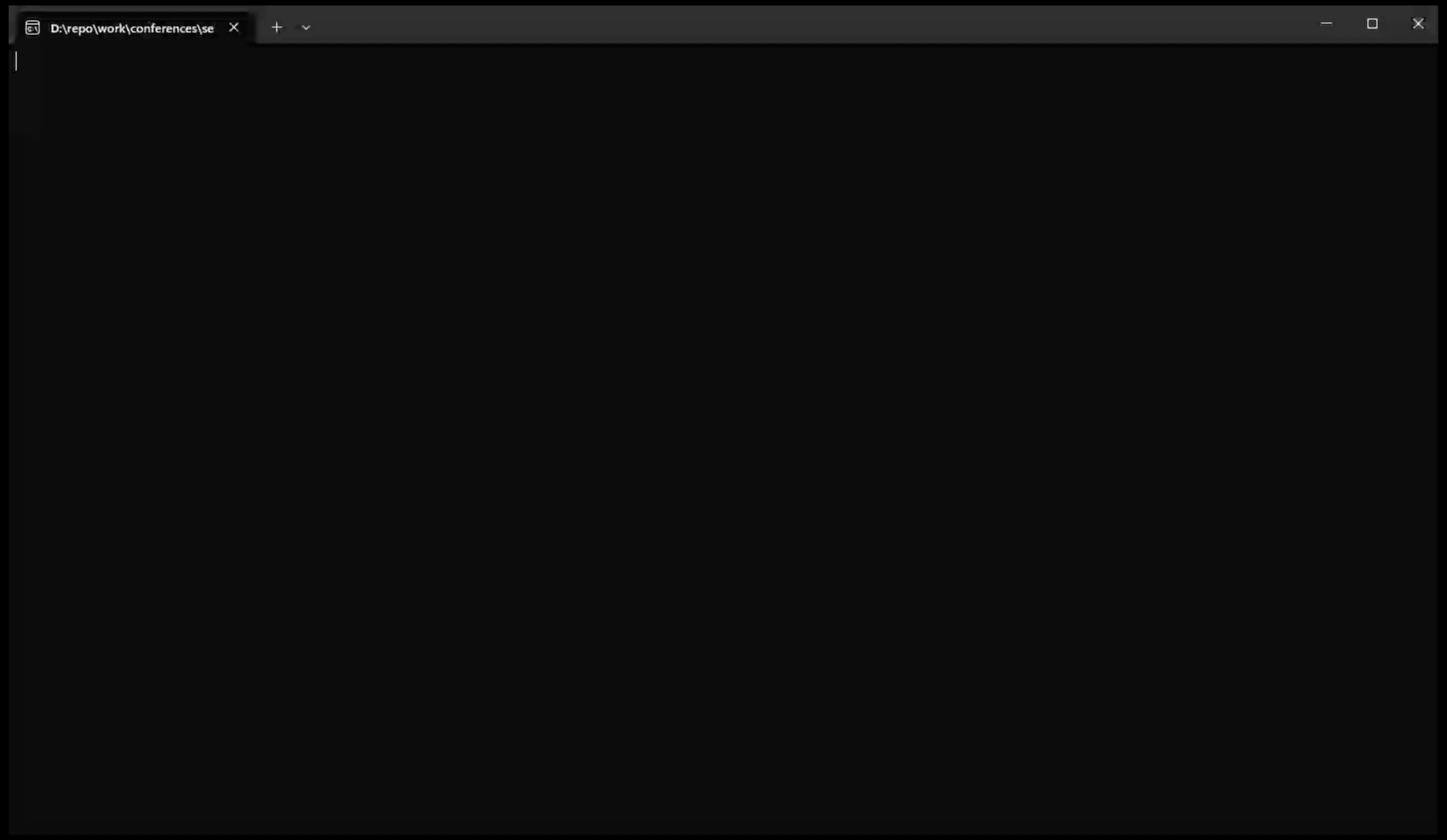
11. Multi AI Models - Setup

```
using Microsoft.Extensions.Configuration;
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.ChatCompletion;
using Microsoft.SemanticKernel.Services;
using Sample;

#pragma warning disable SKEXP0070 // Type is for evaluation purposes only and is subject to change or removal in future upda

Console.WriteLine("=== Kernel AI Model Selection ===");

// 1st GPU Inference AI Model (Ollama - llama3.2)
var ollamaModelId = "llama3.2";
var ollamaEndpoint = new Uri("http://localhost:11434");

// 2nd CPU Inference AI Model (ONNX - phi3)
var onnxModelId = "phi3";
var onnxModelPath = "D:\\repo\\huggingface-models\\Phi-3-mini-4k-instruct-onnx\\cpu_and_mobile\\cpu-int4-rtn-block-32";

// 3rd Cloud Inference AI Model (OpenAI - gpt-4o-mini)
var openAIModelId = "gpt-4o-mini";
var apiKey = new ConfigurationBuilder().AddUserSecrets<Program>().Build()["OpenAI:ApiKey"]!;

var kernelBuilder = Kernel.CreateBuilder();

kernelBuilder
        .AddOpenAIChatCompletion(openAIModelId, apiKey)
        .AddOnnxRuntimeGenAIChatCompletion(onnxModelId, onnxModelPath)
        .AddOllamaChatCompletion(ollamaModelId, ollamaEndpoint);
        // Last service will be the default service

var kernel = kernelBuilder.Build();

kernel.PromptRenderFilters.Add(new SelectedModelRenderFilter());

var modelNames = kernel.GetAllServices<IChatCompletionService>().Select(s => s.GetModelId());
```

```
while (true)
{
    Console.Write($"\nAvailable models: {string.Join(", ", modelNames)}\nModel name > ");
    var selectedModel = Console.ReadLine();
    if (string.IsNullOrEmpty(selectedModel)) { break; }

    if (!modelNames.Contains(selectedModel))
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Invalid model name");
        Console.ResetColor();
        continue;
    }

    Console.Write("User > ");

    var userPrompt = Console.ReadLine();

    if (string.IsNullOrEmpty(userPrompt)) { break; }

    var settings = new PromptExecutionSettings { ModelId = selectedModel };
    var isFirstToken = true;
    await foreach (var token in kernel.InvokePromptStreamingAsync(userPrompt, new(settings)))
    {
        if (isFirstToken)
        {
            Console.Write("\nAssistant > ");
            isFirstToken = false;
        }

        Console.Write(token);
    };
}
```

```
/// <summary>
/// This filter is used to intercept before a kernel prompt is rendered and write to the console what is the selected model in
/// the prompt execution settings.
/// </summary>

public class SelectedModelRenderFilter : IPromptRenderFilter
{
    public async Task OnPromptRenderAsync(PromptRenderContext context, Func<PromptRenderContext, Task> next)
    {
        // For majority of scenarios the prompt execution settings will be a single setting item, for simplicity capturing the f
        Console.WriteLine($"Selected ModelId > {context.Arguments.ExecutionSettings?.FirstOrDefault().Value.ModelId ?? "--"}");

        await next(context);
    }
}
```

11. Multi AI Models - Execution



12. Multi Modalities - Setup

```
using Microsoft.Extensions.Configuration;
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.AudioToText;
using Microsoft.SemanticKernel.ChatCompletion;
using Microsoft.SemanticKernel.Connectors.OpenAI;

#pragma warning disable SKEXP0001 // Type is for evaluation purposes only and is subject to chang
#pragma warning disable SKEXP0010 // Type is for evaluation purposes only and is subject to chang

var apiKey = new ConfigurationBuilder().AddUserSecrets<Program>().Build()["OpenAI:ApiKey"]!;

// Used for audio speech transcription
var audioToTextService = new OpenAIAudioToTextService("whisper-1", apiKey);

// Used for image generation
var textToImageService = new OpenAITextToImageService(modelId: "dall-e-3", apiKey: apiKey);

// Used for image description
var chatService = new OpenAIChatCompletionService("gpt-4o-mini", apiKey);

// Used for audio speech generation
var textToAudioService = new OpenAITextToAudioService("tts-1", apiKey);

Console.WriteLine("""
=== Multi Modality Sample ===
Pipeline:
1. Load the input audio request
2. Transcribe the audio speech into text
3. Use the transcription as the prompt to generate an image
4. Capture a detailed text description from the generated image
5. Generate and audio speech from the detailed image description

Starting ...
""");
```

12. Multi Modalities – Audio-to-Text

Audio-to-Text

“Hey, AI can you generate for me a futuristic metropolis city in 2050, please”

```
35
36      Console.Write($"\n1. Loading audio file ... ");
37      var audioContent = new AudioContent(File.ReadAllBytes("Audios/input.m4a"), "audio/m4a");
38      #pragma warning restore SKEXP0001 // Type is for evaluation purposes only and is subject to change or removal in future updates. Suppress this diagnostic to proceed.
39      Console.WriteLine("Done");
40
41      Console.Write("\n2. Transcribing audio speech into text ... ");
42      var audioTranscriptionText = await audioToTextService.GetTextContentAsync(audioContent);
43      Console.WriteLine("Done\n");
44
45      Console.WriteLine($"Transcription: {audioTranscriptionText}");
46
47      Console.WriteLine("\n3. Generating image from transcription ... ");
48      var generatedImage = (await textToImageService.GetImageContentsAsync(audioTranscriptionText, new OpenAITextToImageExecutionSettings { ResponseFormat = "bytes" })).First();
49      Console.Write("Done\n");
50
51      ChatHistory chatHistory = [];
52      var imageContent = new ImageContent(generatedImage.Data!.Value, "image/jpeg");
53
54      // Save the image to disk
55      var outputImagePath = "Images/output.jpg";
56      if (!Directory.Exists("Images")) { Directory.CreateDirectory("Images"); }
57      if (File.Exists(outputImagePath)) { File.Delete(outputImagePath); }
58      await File.WriteAllBytesAsync(outputImagePath, generatedImage.Data!.Value.ToArray());
59
60      var currentDirectory = Directory.GetCurrentDirectory();
61      Console.WriteLine($"Image Generated. Ctrl + Click to view: {new Uri(Path.Combine(currentDirectory, outputImagePath)).AbsoluteUri}");
62
63      chatHistory.AddUserMessage(
64          [
65              new TextContent("Describe the image in detail"),
66              imageContent
67          ]);
68
69      Console.Write("\n4. Generating image description ... ");
70      var imageDescriptionText = await chatService.GetChatMessageContentAsync(chatHistory);
71      Console.Write("Done\n");
72
73      Console.WriteLine($"Image Description: {imageDescriptionText}");
74
75      Console.Write("\n5. Generating audio speech from image description ... ");
76      var generatedAudio = (await textToAudioService.GetAudioContentsAsync(imageDescriptionText.ToString())).First();
77      Console.Write("Done\n");
78
79      // Save the audio to disk
80      var outputAudioPath = "Audios/output.mp3";
81      if (File.Exists(outputAudioPath)) { File.Delete(outputAudioPath); }
82      await File.WriteAllBytesAsync(outputAudioPath, generatedAudio.Data!.Value.ToArray());
83
84      Console.WriteLine($"Audio speech from image description: (Ctrl + Click to view) {new Uri(Path.Combine(currentDirectory, outputAudioPath)).AbsoluteUri}");
```

12. Multi Modalities – Text-to-Image

```
35
36      Console.Write($"\n1. Loading audio file ... ");
37      var audioContent = new AudioContent(File.ReadAllBytes("Audios/input.m4a"), "audio/m4a");
38      #pragma warning restore SKEXP0001 // Type is for evaluation purposes only and is subject to change or removal in future updates. Suppress this diagnostic to proceed.
39      Console.WriteLine("Done");
40
41      Console.Write("\n2. Transcribing audio speech into text ... ");
42      var audioTranscriptionText = await audioToTextService.GetTextContentAsync(audioContent);
43      Console.WriteLine("Done\n");
44
45      Console.WriteLine($"Transcription: {audioTranscriptionText}");
46
47      Console.WriteLine("\n3. Generating image from transcription ... ");
48      var generatedImage = (await textToImageService.GetImageContentsAsync(audioTranscriptionText, new OpenAITextToImageExecutionSettings { ResponseFormat = "bytes" })).First();
49      Console.Write("Done\n");
50
51      ChatHistory chatHistory = [];
52      var imageContent = new ImageContent(generatedImage.Data!.Value, "image/jpeg");
53
54      // Save the image to disk
55      var outputImagePath = "Images/output.jpg";
56      if (!Directory.Exists("Images")) { Directory.CreateDirectory("Images"); }
57      if (File.Exists(outputImagePath)) { File.Delete(outputImagePath); }
58      await File.WriteAllBytesAsync(outputImagePath, generatedImage.Data!.Value.ToArray());
59
60      var currentDirectory = Directory.GetCurrentDirectory();
61      Console.WriteLine($"Image Generated. Ctrl + Click to view: {new Uri(Path.Combine(currentDirectory, outputImagePath)).AbsoluteUri}");
62
63      chatHistory.AddUserMessage(
64          [
65              new TextContent("Describe the image in detail"),
66              imageContent
67          ]);
68
69      Console.Write("\n4. Generating image description ... ");
70      var imageDescriptionText = await chatService.GetChatMessageContentAsync(chatHistory);
71      Console.Write("Done\n");
72
73      Console.WriteLine($"Image Description: {imageDescriptionText}");
74
75      Console.Write("\n5. Generating audio speech from image description ... ");
76      var generatedAudio = (await textToAudioService.GetAudioContentsAsync(imageDescriptionText.ToString())).First();
77      Console.Write("Done\n");
78
79      // Save the audio to disk
80      var outputAudioPath = "Audios/output.mp3";
81      if (File.Exists(outputAudioPath)) { File.Delete(outputAudioPath); }
82      await File.WriteAllBytesAsync(outputAudioPath, generatedAudio.Data!.Value.ToArray());
83
84      Console.WriteLine($"Audio speech from image description: (Ctrl + Click to view) {new Uri(Path.Combine(currentDirectory, outputAudioPath)).AbsoluteUri}");
```

Audio-to-Text

“Hey, AI can you generate for me a futuristic metropolis city in 2050, please”

Text-to-Image

12. Multi Modalities – Chat-Image-to-Text

```
35
36      Console.Write($"\n1. Loading audio file ... ");
37      var audioContent = new AudioContent(File.ReadAllBytes("Audios/input.m4a"), "audio/m4a");
38      #pragma warning restore SKEXP0001 // Type is for evaluation purposes only and is subject to change or removal in future updates. Suppress this diagnostic to proceed.
39      Console.WriteLine("Done");
40
41      Console.Write("\n2. Transcribing audio speech into text ... ");
42      var audioTranscriptionText = await audioToTextService.GetTextContentAsync(audioContent);
43      Console.WriteLine("Done\n");
44
45      Console.WriteLine($"Transcription: {audioTranscriptionText}");
46
47      Console.WriteLine("\n3. Generating image from transcription ... ");
48      var generatedImage = (await textToImageService.GetImageContentsAsync(audioTranscriptionText, new OpenAITextToImageExecutionSettings { ResponseFormat = "bytes" })).First();
49      Console.Write("Done\n");
50
51      ChatHistory chatHistory = [];
52      var imageContent = new ImageContent(generatedImage.Data!.Value, "image/jpeg");
53
54      // Save the image to disk
55      var outputImagePath = "Images/output.jpg";
56      if (!Directory.Exists("Images")) { Directory.CreateDirectory("Images"); }
57      if (File.Exists(outputImagePath)) { File.Delete(outputImagePath); }
58      await File.WriteAllBytesAsync(outputImagePath, generatedImage.Data!.Value.ToArray());
59
60      var currentDirectory = Directory.GetCurrentDirectory();
61      Console.WriteLine($"Image Generated. Ctrl + Click to view: {new Uri(Path.Combine(currentDirectory, outputImagePath)).AbsoluteUri}");
62
63      chatHistory.AddUserMessage(
64          [
65              new TextContent("Describe the image in detail"),
66              imageContent
67          ]);
68
69      Console.Write("\n4. Generating image description ... ");
70      var imageDescriptionText = await chatService.GetChatMessageContentAsync(chatHistory);
71      Console.Write("Done\n");
72
73      Console.WriteLine($"Image Description: {imageDescriptionText}");
74
75      Console.Write("\n5. Generating audio speech from image description ... ");
76      var generatedAudio = (await textToAudioService.GetAudioContentsAsync(imageDescriptionText.ToString())).First();
77      Console.Write("Done\n");
78
79      // Save the audio to disk
80      var outputAudioPath = "Audios/output.mp3";
81      if (File.Exists(outputAudioPath)) { File.Delete(outputAudioPath); }
82      await File.WriteAllBytesAsync(outputAudioPath, generatedAudio.Data!.Value.ToArray());
83
84      Console.WriteLine($"Audio speech from image description: (Ctrl + Click to view) {new Uri(Path.Combine(currentDirectory, outputAudioPath)).AbsoluteUri}");
```

Audio-to-Text

“Hey, AI can you generate for me a futuristic metropolis city in 2050, please”

Text-to-Image

Chat-Image-to-Text

“The image depicts a futuristic cityscape... ... that enhance connectivity and livability.”

12. Multi Modalities – Text-to-Audio

```
35
36      Console.Write($"\n1. Loading audio file ... ");
37      var audioContent = new AudioContent(File.ReadAllBytes("Audios/input.m4a"), "audio/m4a");
38      #pragma warning restore SKEXP0001 // Type is for evaluation purposes only and is subject to change or removal in future updates. Suppress this diagnostic to proceed.
39      Console.WriteLine("Done");
40
41      Console.Write("\n2. Transcribing audio speech into text ... ");
42      var audioTranscriptionText = await audioToTextService.GetTextContentAsync(audioContent);
43      Console.WriteLine("Done\n");
44
45      Console.WriteLine($"Transcription: {audioTranscriptionText}");
46
47      Console.WriteLine("\n3. Generating image from transcription ... ");
48      var generatedImage = (await textToImageService.GetImageContentsAsync(audioTranscriptionText, new OpenAITextToImageExecutionSettings { ResponseFormat = "bytes" })).First();
49      Console.Write("Done\n");
50
51      ChatHistory chatHistory = [];
52      var imageContent = new ImageContent(generatedImage.Data!.Value, "image/jpeg");
53
54      // Save the image to disk
55      var outputImagePath = "Images/output.jpg";
56      if (!Directory.Exists("Images")) { Directory.CreateDirectory("Images"); }
57      if (File.Exists(outputImagePath)) { File.Delete(outputImagePath); }
58      await File.WriteAllBytesAsync(outputImagePath, generatedImage.Data!.Value.ToArray());
59
60      var currentDirectory = Directory.GetCurrentDirectory();
61      Console.WriteLine($"Image Generated. Ctrl + Click to view: {new Uri(Path.Combine(currentDirectory, outputImagePath)).AbsoluteUri}");
62
63      chatHistory.AddUserMessage(
64          [
65              new TextContent("Describe the image in detail"),
66              imageContent
67          ]);
68
69      Console.Write("\n4. Generating image description ... ");
70      var imageDescriptionText = await chatService.GetChatMessageContentAsync(chatHistory);
71      Console.Write("Done\n");
72
73      Console.WriteLine($"Image Description: {imageDescriptionText}");
74
75      Console.Write("\n5. Generating audio speech from image description ... ");
76      var generatedAudio = (await textToAudioService.GetAudioContentsAsync(imageDescriptionText.ToString())).First();
77      Console.Write("Done\n");
78
79      // Save the audio to disk
80      var outputAudioPath = "Audios/output.mp3";
81      if (File.Exists(outputAudioPath)) { File.Delete(outputAudioPath); }
82      await File.WriteAllBytesAsync(outputAudioPath, generatedAudio.Data!.Value.ToArray());
83
84      Console.WriteLine($"Audio speech from image description: (Ctrl + Click to view) {new Uri(Path.Combine(currentDirectory, outputAudioPath)).AbsoluteUri}");
```

Audio-to-Text

“Hey, AI can you generate for me a futuristic metropolis city in 2050, please”

Text-to-Image

Chat-Image-to-Text

“The image depicts a futuristic cityscape... ... that enhance connectivity and livability.”

Text-to-Audio

12. Multi Modalities - Execution

Audio-to-Text

“Hey, AI can you generate for me a futuristic metropolis city in 2050, please”

Text-to-Image

Chat-Image-to-Text

“The image depicts a futuristic cityscape...
... that enhance connectivity and livability.”

Text-to-Audio

```
C:\Users\roger\OneDrive\Doc
=== Multi Modality Sample ===
Pipeline:
1. Load the input audio request
2. Transcribe the audio speech into text
3. Use the transcription as the prompt to generate an image
4. Capture a detailed text description from the generated image
5. Generate and audio speech from the detailed image description

Starting ...
1. Loading audio file ... Done
2. Transcribing audio speech into text ... Done
Transcription: Hey, I can generate for me a futuristic metropolis city in 2050, please
3. Generating image from transcription ...
Done
Image Generated. Ctrl + Click to view: file:///D:/repo/work/conferences/semantic-kernel-demos/Demos/Beginner-Hands-On/Samples/MultiModality/Images/output.jpg
4. Generating image description ... Done
Image Description: The image depicts a futuristic cityscape brimming with advanced technology and architecture. Towering skyscrapers made of glass and metal dominate the skyline, reflecting a vibrant sunset in the background. The buildings are sleek and modern, featuring various designs, from cylindrical forms to angular structures, many adorned with glowing blue and purple lights.

In the foreground, a circular plaza acts as a focal point, surrounded by lush greenery and modern architecture. High-tech transportation systems are visible, with roadways marked by glowing lines and hexagonal shapes indicating lanes or traffic control elements. Roads are bustling with vehicles, including autonomous cars that seem to glide effortlessly along the streets.

Above the city, drones and flying vehicles navigate through the air, contributing to a sense of dynamic movement and innovation. Holographic displays can be seen projected in various areas, further emphasizing the advanced technological theme of the city.

The overall atmosphere is one of progress and efficiency, showcasing a potential vision of urban life infused with futuristic elements that enhance connectivity and livability.
5. Generating audio speech from image description ... Done
Image Description Audio: (Ctrl + Click to view) file:///D:/repo/work/conferences/semantic-kernel-demos/Demos/Beginner-Hands-On/Samples/MultiModality/Audios/output.mp3
```

Support Channels for Semantic Kernel

1. Discord – Community driven
2. Office Hours – weekly open meeting for anyone to join and ask questions of the product team (General Questions)
3. GitHub Issues – reviewed daily (Feature / Bugs)
4. Microsoft Support – open issue that is routed to the team (Urgent)

Get Involved

Join the community

Office hours every Wednesday at 8 AM PST - https://aka.ms/sk-community-calendar

Microsoft Learn

MS Learn
https://aka.ms/sk/learn

DevBlogs

SK DevBlogs
https://aka.ms/sk/devblogs

Office hours every Wednesday at 8 AM PST - https://aka.ms/sk-community-calendar

Questions?

Office hours every Wednesday at 8 AM PST

https://aka.ms/sk-community-calendar