Semantic Kernel: A Comprehensive Guide from Beginner to Advanced

1. Introduction to Semantic Kernel

Microsoft Semantic Kernel (SK) emerges as a pivotal open-source Software Development Kit (SDK) designed to seamlessly integrate the power of advanced Artificial Intelligence (AI) Large Language Models (LLMs) with traditional programming languages such as C#, Python, and Java.[1] Its fundamental purpose is to empower developers to construct robust and future-proof AI solutions that can adapt and evolve alongside rapid technological advancements in the field.[5] This is achieved by providing a framework that allows for the fluid blending of cutting-edge AI capabilities with established native code functionalities.[6]

Serving as an efficient middleware, Semantic Kernel streamlines the process of delivering enterprise-grade AI solutions with speed and agility.[4] Its core functionalities are multifaceted, encompassing the ability to build sophisticated AI Agents [4], seamlessly integrate a wide array of AI Models [4], and automate intricate business processes by intelligently combining natural language prompts with existing application programming interfaces (APIs).[4] By enabling the description of conventional code to AI models, Semantic Kernel allows these models to invoke and utilize that code to address specific requests, effectively acting as a translator between the AI's understanding and the execution of programmatic functions.[4]

One of the defining characteristics of Semantic Kernel is its model-agnostic nature, offering built-in support for a diverse range of LLMs from prominent providers like OpenAI, Azure OpenAI, Hugging Face, and NVidia, among others.[3] This design ensures that developers are not locked into a single AI ecosystem and can easily swap out underlying models as new advancements become available, without necessitating a complete rewrite of their application's core logic.[3]

At its heart, Semantic Kernel can be conceptualized as an AI orchestration layer, much like the sophisticated systems that power Microsoft's own intelligent products such as Microsoft 365 Copilot and Bing.[1] By providing developers with access to these same underlying AI orchestration patterns, Semantic Kernel enables them to build their own innovative Copilot-like experiences within their applications, all while leveraging their existing development skills and investments.[1]

The recurring emphasis on the terms "integration" and "orchestration" throughout the documentation [1] underscores a central theme: Semantic Kernel's primary strength lies in its capacity to function as a unified platform for combining a multitude of AI functionalities with traditional programming practices. This suggests a development philosophy centered on creating complex AI applications through the modular assembly and coordinated management of various specialized components. Furthermore, the fact that Semantic Kernel underpins Microsoft's Copilot system [1] lends significant credibility to the SDK, indicating that it is built upon robust, production-validated principles employed in large-scale AI deployments.

Initial Setup and Installation

Semantic Kernel offers broad accessibility by supporting multiple widely-used programming languages, including C#, Python, and Java.[3] The installation process is straightforward for each language:

- **For.NET (C#):** Developers can easily add the Semantic Kernel library to their projects using the NuGet package manager with the command: dotnet add package Microsoft.SemanticKernel.[7]
- **For Python:** The SDK can be installed using the pip package installer with the command: pip install semantic-kernel.[7]
- **For Java:** Integration into Java projects requires adding the appropriate dependencies to the project's pom.xml file, as detailed in the official documentation.[9]

For those looking for a rapid introduction and experimentation, the documentation recommends utilizing notebooks, such as Jupyter notebooks for Python and VS Code notebooks for.NET.[9] These interactive environments allow for step-by-step execution and immediate feedback. To get started with notebooks, developers can clone the official Semantic Kernel repository from GitHub and open the relevant getting-started notebook located within the repository.[9]

The initial code structure for initializing the Kernel, the central orchestrator in Semantic Kernel, is concise across the supported languages:

- **Python:**
  Python
  ```python
  from semantic_kernel import Kernel
  kernel = Kernel()
  ```
  This simple instantiation creates a new Kernel object, ready to be configured with AI services and plugins.[9]
- **.NET (C#):**
  C#
  ```csharp
  using Microsoft.SemanticKernel;
  ```

```
var kernel = Kernel.CreateBuilder().Build();
```
In.NET, the Kernel is typically created using a builder pattern, allowing for a fluent configuration of services and plugins before the Kernel instance is finalized.[7]

- **Java:**

  ```Java
  import com.microsoft.semantickernel.Kernel;

  Kernel kernel = Kernel.builder().build();
  ```
  Similar to.NET, Java utilizes a builder pattern for constructing the Kernel, providing a structured approach to adding necessary components.[5]

The availability of the SDK in these prominent programming languages signifies a strategic decision to make Semantic Kernel accessible to a broad spectrum of developers, catering to diverse technical backgrounds and development environments.

2. Core Concepts of Semantic Kernel

The Kernel

At the core of the Semantic Kernel framework lies the **Kernel**, which serves as the central orchestrator and runtime environment for all AI operations.[5] It acts as a sophisticated **Dependency Injection (DI) container**, responsible for managing the lifecycle and dependencies of various services and plugins that constitute an AI application built with Semantic Kernel.[5] This design principle promotes modularity and allows different components of the system to interact without tight coupling.

The Kernel plays a pivotal role in the execution of AI tasks, particularly when invoking prompts. Its responsibilities include intelligently selecting the most appropriate AI service to handle a given prompt, constructing the prompt by utilizing predefined prompt templates, dispatching the prompt to the chosen AI service, and subsequently receiving and parsing the response generated by the service. Finally, the Kernel returns this response back to the calling application.[5]

Throughout the execution of prompts and code within a Semantic Kernel application, the Kernel remains readily available to retrieve any necessary services and plugins that are required for the task at hand.[5] This centralized management capability provides developers with a unified point for both configuring and monitoring their AI agents, offering significant control and observability over the system's behavior.[5]

The Kernel fundamentally manages two distinct types of components:

- **Services:** This category encompasses both the core AI services, such as chat completion and text generation, as well as other essential utilities needed for running the application, including logging frameworks and HTTP clients. The design of service management within the Kernel follows the Service Provider pattern commonly found in.NET, ensuring robust dependency injection support across all supported programming languages.[5]
- **Plugins:** These are the functional units that AI services and prompt templates leverage to perform specific tasks. For instance, an AI service might utilize a plugin to fetch real-time data from a database or interact with external APIs to carry out a particular action.[5]

To illustrate the creation and configuration of the Kernel, consider the following code examples:

C#:

C#

```
using Microsoft.Extensions.Logging;
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Connectors.OpenAI;

// Configure logging
using ILoggerFactory loggerFactory = LoggerFactory.Create(builder =>
{
    builder.AddConsole();
    builder.SetMinimumLevel(LogLevel.Information);
});

// Create a Kernel builder
var builder = Kernel.CreateBuilder();

// Add Azure OpenAI chat completion service
builder.AddAzureOpenAIChatCompletion(
    deploymentName: "your-deployment-name",
    endpoint: "your-endpoint",
    apiKey: "your-api-key");

// Add a logger
builder.Services.AddSingleton(loggerFactory);

// Build the Kernel
Kernel kernel = builder.Build();
```

This example demonstrates how to create a Kernel builder, add an Azure OpenAI chat completion service with necessary credentials, configure logging to the console, and finally build the Kernel instance.[5]

Python:

Python

```python
import asyncio
import logging
from semantic_kernel import Kernel
from semantic_kernel.connectors.ai.open_ai import AzureChatCompletion
from semantic_kernel.utils.logging import setup_logging

# Setup logging
setup_logging()
logging.getLogger('semantic_kernel').setLevel(logging.DEBUG)

async def main():
    # Initialize the Kernel
    kernel = Kernel()

    # Add Azure OpenAI chat completion service
    kernel.add_service(AzureChatCompletion(
        service_id="azure_openai",
        deployment_name="your-deployment-name",
        endpoint="your-endpoint",
        api_key="your-api-key"
    ))
```

```python
if __name__ == "__main__":
    asyncio.run(main())
```

This Python example showcases the initialization of a Kernel and the addition of an Azure OpenAI chat completion service, along with basic logging configuration.[5]

Java:

```java
Java

import com.azure.core.credential.AzureKeyCredential;
import com.azure.openai.OpenAIAsyncClient;
import com.azure.openai.OpenAIClientBuilder;
import com.microsoft.semantickernel.Kernel;
import
com.microsoft.semantickernel.aiservices.openai.chatcompletion.OpenAIChatCompleti
on;
import com.microsoft.semantickernel.plugin.KernelPlugin;
import com.microsoft.semantickernel.plugin.functional.KernelPluginFactory;

public class ExampleKernel {
    public static void main(String args) {
        String apiKey = "your-api-key";
        String deploymentName = "your-deployment-name";
        String endpoint = "your-endpoint";
        String modelId = "your-model-id";

        OpenAIAsyncClient client = new OpenAIClientBuilder()
                .credential(new AzureKeyCredential(apiKey))
                .endpoint(endpoint)
                .buildAsyncClient();

        KernelPlugin timePlugin = KernelPluginFactory.createFromObject(new
TimePlugin(), "TimePlugin");

        OpenAIChatCompletion chatCompletionService =
OpenAIChatCompletion.builder()
            .withModelId(modelId)
            .withOpenAIAsyncClient(client)
            .build();

      Kernel kernel = Kernel.builder()
            .withAIService(OpenAIChatCompletion.class, chatCompletionService)
            .withPlugin(timePlugin)
            .build();
   }
}

// A simple native plugin example in Java
class TimePlugin {
   @com.microsoft.semantickernel.plugin.annotations.KernelFunction()
   public String getCurrentTime() {
      return java.time.LocalTime.now().toString();
   }
}
```

This Java example illustrates building a Kernel, adding an OpenAI chat completion service, and incorporating a simple native plugin.[5]

The Kernel's role as a dependency injection container [5] signifies an architectural commitment to modularity, maintainability, and testability. This design allows developers to easily swap out or extend the functionality of different components within the Semantic Kernel framework without creating tight dependencies between them.

Plugins (Skills)

A **plugin**, often referred to as a skill, represents a collection of functions, either implemented in native code or defined as semantic prompts, that can be exposed to AI services and applications.[6] These plugins serve as the fundamental building blocks for creating AI applications with Semantic Kernel.[13] By encapsulating specific functionalities, plugins enable the Kernel to perform a wide range of tasks, from retrieving information to automating complex workflows.[12]

Plugins can include both native code and natural language prompts, allowing developers to leverage the power of generative AI within their applications.[13] This flexibility enables the definition of desired application behaviors, and developers can create custom prompt plugins to fine-tune their applications precisely to their needs.[13] Within a plugin, functions are typically categorized into those that retrieve data for **Retrieval Augmented Generation (RAG)** and those that automate tasks.[12]

The process of utilizing plugins within Semantic Kernel generally involves three key steps: first, defining the plugin and its functions; second, adding the plugin to the Kernel instance; and third, invoking the desired function from the plugin, either directly within a prompt or through function calling mechanisms.[12]

Semantic Functions

Semantic Functions are essentially prompts that are executed against a Large Language Model (LLM) to generate natural language responses.[6] These functions rely on connectors to interact with the underlying AI models and fulfill their purpose.[6] The creation of semantic functions often involves the use of custom prompt templates, which allow for the dynamic insertion of variables and the execution of other functions within the prompt itself.[17] This templating capability enables the generation of tailored prompts based on specific user inputs or contextual information, making them highly versatile for a wide array of applications.[17]

Configuration details for semantic functions are typically defined in a config.json file that accompanies the prompt template. This configuration file supports various parameters, including the function's name, a concise description of its purpose, definitions of input and output parameters, and execution settings that govern how the LLM should process the prompt.[13]

Semantic functions can be invoked by using a ContextVariables object to pass data as parameters to the function. This allows developers to dynamically control the behavior and output of the semantic function based on the specific context of the application.[6]

Consider the following code examples illustrating the creation and invocation of semantic functions:

C#:

C#

```
using Microsoft.SemanticKernel;
```

```csharp
using Microsoft.SemanticKernel.PromptTemplates.Handlebars;

// Assuming a Kernel instance 'kernel' is already initialized

// Define a prompt template using Handlebars syntax
string promptTemplate = @"
Today is {{date}}. Please tell me an interesting fact about this date in history.
Be sure to mention the year in your response.
";

// Create a Handlebars prompt template factory
var templateFactory = new HandlebarsPromptTemplateFactory();

// Create a semantic function from the prompt template
var dailyFactFunction = kernel.CreateFunctionFromPrompt(
    promptTemplate: promptTemplate,
    promptTemplateConfig: new PromptTemplateConfig()
    {
        Name = "GetDailyFact",
        Description = "Provides an interesting historical fact for a given date."
    },
    templateFactory: templateFactory);

// Invoke the semantic function
KernelResult result = await kernel.InvokeAsync(dailyFactFunction, new
KernelArguments() { ["date"] = DateTime.Now.ToString("MMMM dd") });

Console.WriteLine(result.GetValue<string>());
```

This C# example demonstrates creating a semantic function using a Handlebars prompt template that includes a date variable. The function is then invoked with the current date, and the result (an interesting historical fact) is printed to the console.[17]

Python:

Python

```python
import asyncio
```

```
from semantic_kernel import Kernel
from semantic_kernel.functions import kernel_function, KernelFunctionFromPrompt
from semantic_kernel.connectors.ai.open_ai import
OpenAIChatPromptExecutionSettings

async def main():
    kernel = Kernel()

    # Add a chat service (replace with your actual configuration)
    kernel.add_service(AzureChatCompletion(service_id="azure_openai",
deployment_name="your-deployment-name", endpoint="your-endpoint",
api_key="your-api-key"))

    # Define a prompt for a semantic function
    prompt = """{{$input}}
   Translate the above text to French.
   """;

    # Create execution settings
    execution_settings = OpenAIChatPromptExecutionSettings(
        service_id="azure_openai",
        temperature=0.7,
        max_tokens=200
    )

    # Create a semantic function from the prompt
    translate_function = KernelFunctionFromPrompt(
        prompt=prompt,
        execution_settings=execution_settings,
        name="TranslateToFrench",
        description="Translates text to French."
    )

    # Invoke the semantic function
    result = await kernel.invoke(translate_function, input="Hello, how are you?")
    print(f"Translated text: {result}")

if __name__ == "__main__":
    asyncio.run(main())
```

This Python example shows how to define a semantic function using KernelFunctionFromPrompt with a prompt that includes an input variable. The function is then invoked with the input text "Hello, how are you?", and the French translation is printed.[18]

It is crucial to employ descriptive and concise function names and clearly named parameters for semantic functions. This practice significantly enhances the LLM's ability to understand the purpose of each function and when it is most appropriate to utilize them.[12] Furthermore, keeping prompt templates simple and well-organized is essential for maintaining readability and facilitating easier debugging.[17]

Native Functions

Native Functions are code functions written in conventional programming languages such as C#, Python, or Java. These functions are designed to handle operations that might not be best suited for AI models alone, such as performing complex mathematical calculations, reading and writing data to memory or databases, or interacting with external REST APIs.[6]

To expose native functions to the Semantic Kernel, developers need to use specific decorators or attributes provided by the SDK. In C#, the [KernelFunction] attribute, often accompanied by the [Description] attribute for providing semantic context, is used to mark a method as a native function that the Kernel can discover and invoke.[6] Similarly, in Python, the @kernel_function decorator serves the same purpose, allowing developers to annotate their Python functions with descriptions and parameter information.[18] While Java support is evolving, similar annotation mechanisms are employed to identify native functions within Java plugins.

These semantic descriptions are vital as they provide the AI agent with the necessary information to understand the capabilities of the plugin and how to interact with its functions.[6] This includes the name of the plugin, the names of the functions it contains, detailed descriptions of what each function does, the parameters it accepts, and the schema of those parameters and the return value.[20]

Here are some code examples demonstrating the creation and registration of native functions:

C#:

C#

```csharp
using Microsoft.SemanticKernel;
using System.ComponentModel;

public class TimePlugin
{
    [KernelFunction, Description("Gets the current day of the week.")]
    public string GetDayOfWeek()
    {
        return DateTime.Now.DayOfWeek.ToString();
    }

    [KernelFunction, Description("Gets the current time.")]
    public string GetCurrentTime()
    {
        return DateTime.Now.ToString("HH:mm:ss");
    }
}

// In your main application code:
// Assuming a Kernel instance 'kernel' is already initialized
kernel.ImportPluginFromObject(new TimePlugin(), "TimePlugin");
```

In this C# example, a TimePlugin class is defined with two native functions, GetDayOfWeek and GetCurrentTime, both decorated with [KernelFunction] and [Description] attributes. The plugin is then imported into the Kernel using ImportPluginFromObject.[19]

Python:

Python

```python
from semantic_kernel import Kernel
from semantic_kernel.functions import kernel_function
import datetime
```

```python
class CalendarPlugin:
    @kernel_function(description="Get the current date")
    def get_date(self) -> str:
        """Returns the current date."""
        return datetime.date.today().strftime("%Y-%m-%d")

    @kernel_function(description="Get the current time")
    def get_time(self) -> str:
        """Returns the current time."""
        return datetime.datetime.now().strftime("%H:%M:%S")

async def main():
    kernel = Kernel()
    # Add a chat service (replace with your actual configuration)
    kernel.add_service(AzureChatCompletion(service_id="azure_openai",
deployment_name="your-deployment-name", endpoint="your-endpoint",
api_key="your-api-key"))

    # Create an instance of the CalendarPlugin
    calendar_plugin = CalendarPlugin()

    # Add the plugin to the kernel
    kernel.add_plugin(calendar_plugin, plugin_name="CalendarPlugin")

if __name__ == "__main__":
    asyncio.run(main())
```

This Python example defines a CalendarPlugin class with two native functions, get_date and get_time, both decorated with @kernel_function and a description. An instance of the plugin is then added to the Kernel using kernel.add_plugin.[18]

Native functions provide a crucial capability by allowing AI agents built with Semantic Kernel to interact with the real world through the execution of code. This enables the creation of hybrid AI applications that can seamlessly combine the natural language understanding and generation capabilities of LLMs with the procedural execution power of traditional programming.

Connectors

Connectors in Semantic Kernel act as essential bridges that facilitate communication between the Kernel and external AI or Memory services.[3] They play a vital role in enabling the exchange of information between different components of an AI application, allowing Semantic Kernel to leverage the capabilities of various AI models and memory storage solutions.[6] Connectors can be specifically developed to interact with a wide range of external systems, including models from HuggingFace and memory solutions like SQLite databases.[6]

A key benefit of using connectors is that they simplify the process of integrating with diverse databases and AI services. This abstraction allows developers to concentrate on the core logic of their applications rather than getting bogged down in the intricate details of service-specific configurations.[25] The connector architecture is designed to support multiple types of databases and AI services, making Semantic Kernel highly adaptable to various use cases and development environments.[25]

AI Service Connectors

AI Service Connectors provide a crucial abstraction layer that enables Semantic Kernel to interact with different types of AI services, such as Chat Completion, Text Generation, Embedding Generation, and more, from a variety of providers.[1] These providers include major players like OpenAI, Azure OpenAI, Hugging Face, Google, Mistral, Meta, and NVidia, as well as support for local models.[3] By offering a common interface to these diverse services, Semantic Kernel simplifies the process of leveraging their unique capabilities within an application.[32] When a specific AI service implementation is registered with the Kernel, it becomes the default service for the corresponding type (e.g., Chat Completion or Text Generation) for any subsequent calls made by the Kernel.[32]

One of the significant advantages of AI Service Connectors is the ease with which developers can switch between different AI providers. This flexibility allows for experimentation with various models and their performance characteristics to identify the one that best suits the specific requirements of a given use case.[26]

Here are code examples demonstrating how to add different AI service connectors to the Kernel:

C#:

C#

```csharp
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Connectors.OpenAI;

// Assuming a Kernel builder 'builder' is already initialized

// Add Azure OpenAI chat completion service
builder.AddAzureOpenAIChatCompletion(
    deploymentName: "your-azure-deployment-name",
    endpoint: "your-azure-endpoint",
    apiKey: "your-azure-api-key");

// Add OpenAI chat completion service
builder.AddOpenAIChatCompletion(
    modelId: "gpt-3.5-turbo",
    apiKey: "your-openai-api-key");

// Build the Kernel
Kernel kernel = builder.Build();
```

This C# example shows how to add both Azure OpenAI and OpenAI chat completion services to the Kernel builder, allowing the application to utilize models from either provider.[7]

Python:

```python
Python

import asyncio
from semantic_kernel import Kernel
from semantic_kernel.connectors.ai.open_ai import AzureChatCompletion,
OpenAIChatCompletion
from semantic_kernel.connectors.ai.hugging_face import HuggingFaceTextCompletion

async def main():
    kernel = Kernel()

    # Add Azure OpenAI chat completion service
    kernel.add_service(AzureChatCompletion(
        service_id="azure_openai",
        deployment_name="your-azure-deployment-name",
        endpoint="your-azure-endpoint",
        api_key="your-azure-api-key"
    ))

    # Add OpenAI chat completion service
    kernel.add_service(OpenAIChatCompletion(
        service_id="openai",
        model_id="gpt-3.5-turbo",
        api_key="your-openai-api-key"
    ))

    # Add Hugging Face text completion service
    kernel.add_service(HuggingFaceTextCompletion(
        service_id="huggingface",
        ai_model_id="gpt2",
        task="text-generation"
    ))

if __name__ == "__main__":
    asyncio.run(main())
```

This Python example demonstrates adding Azure OpenAI, OpenAI, and Hugging Face text completion services to the Kernel, showcasing the ability to integrate with multiple AI providers.[7]

Java:

Java

```java
import com.azure.core.credential.AzureKeyCredential;
import com.azure.openai.OpenAIAsyncClient;
import com.azure.openai.OpenAIClientBuilder;
import com.microsoft.semantickernel.Kernel;
import
```

```
com.microsoft.semantickernel.aiservices.openai.chatcompletion.OpenAIChatCompleti
on;

public class ExampleAIServiceConnectors {
  public static void main(String args) {
    String azureApiKey = "your-azure-api-key";
    String azureDeploymentName = "your-azure-deployment-name";
    String azureEndpoint = "your-azure-endpoint";
    String openAiApiKey = "your-openai-api-key";
    String openAiModelId = "gpt-3.5-turbo";

    Kernel kernel = Kernel.builder()
        // Add Azure OpenAI chat completion service
        .withAIService(OpenAIChatCompletion.class, new
OpenAIChatCompletion.Builder()
            .withModelId(azureDeploymentName)
            .withOpenAIAsyncClient(new OpenAIClientBuilder()
                .credential(new AzureKeyCredential(azureApiKey))
                .endpoint(azureEndpoint)
                .buildAsyncClient())
            .build(), "azure_openai")
        // Add OpenAI chat completion service
        .withAIService(OpenAIChatCompletion.class, new
OpenAIChatCompletion.Builder()
            .withModelId(openAiModelId)
            .withOpenAIAsyncClient(new OpenAIClientBuilder()
                .credential(new AzureKeyCredential(openAiApiKey))
                .buildAsyncClient())
            .build(), "openai")
        .build();
  }
}
```

This Java example shows how to add both Azure OpenAI and OpenAI chat completion services to the Kernel builder, similar to the C# example.[7]

The wide range of supported AI service providers through these connectors [1] highlights Semantic Kernel's commitment to model flexibility. This empowers developers to select the most suitable AI model for their specific application requirements and to easily adapt to new advancements in the field.

Memory Connectors

Memory Connectors in Semantic Kernel are responsible for facilitating the interaction between the Kernel and various memory stores, including popular vector databases such as Azure AI Search, Chroma, and Milvus.[6] These connectors abstract away the underlying complexities of communicating with different memory storage solutions, allowing developers to work with a consistent set of APIs.[25] While some memory connectors are currently in an experimental phase, the goal is to refine them to effectively meet the needs of developers working with semantic memory.[25]

Memory connectors simplify the process of connecting to various databases, enabling developers to focus on building their applications rather than dealing with intricate database configurations.[25] The architecture's support for multiple database types ensures adaptability to a wide range of use cases and deployment environments.[25] Semantic Kernel provides an abstraction layer for interacting with Vector Stores, along with a collection of out-of-the-box connectors that implement these abstractions.[35] These connectors offer functionalities for creating, listing, and deleting collections of records, as well as for uploading, retrieving, and deleting individual records.[35]

Here are code examples demonstrating the use of memory connectors:

C#:

C#

```csharp
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Memory;
using Microsoft.SemanticKernel.Connectors.Memory.Volatile;
using Microsoft.SemanticKernel.TextEmbedding;
using Microsoft.SemanticKernel.Connectors.OpenAI;

// Assuming a Kernel builder 'builder' and OpenAI configuration

// Add OpenAI text embedding generation service
builder.Services.AddSingleton<ITextEmbeddingGenerationService>(new
OpenAITextEmbeddingGenerationService(
    modelId: "text-embedding-ada-002",
    apiKey: "your-openai-api-key"));

// Add volatile memory store
builder.Services.AddSingleton<IMemoryStore>(new VolatileMemoryStore());

// Build the Kernel
Kernel kernel = builder.Build();

// Get the memory store
var memoryStore = kernel.GetService<IMemoryStore>();
```

This C# example demonstrates how to configure a Kernel to use volatile memory by adding the VolatileMemoryStore to the service collection and an OpenAI text embedding generation service.[36]

Python:

```
Python

import asyncio
from semantic_kernel import Kernel
from semantic_kernel.memory.volatile_memory_store import VolatileMemoryStore
from semantic_kernel.memory.semantic_text_memory import SemanticTextMemory
from semantic_kernel.connectors.ai.open_ai import OpenAITextEmbedding,
AzureChatCompletion

async def main():
    kernel = Kernel()
    # Add a chat service (replace with your actual configuration)
    kernel.add_service(AzureChatCompletion(service_id="azure_openai",
deployment_name="your-deployment-name", endpoint="your-endpoint",
api_key="your-api-key"))

    # Add OpenAI text embedding service
    kernel.add_service(OpenAITextEmbedding(service_id="openai_embedding",
model_id="text-embedding-ada-002", api_key="your-openai-api-key"))
```

```
    # Use volatile memory store
    memory_store = VolatileMemoryStore()
    kernel.memory = SemanticTextMemory(storage=memory_store,
embeddings_generator=kernel.get_service(OpenAITextEmbedding))

if __name__ == "__main__":
    asyncio.run(main())
```

This Python example shows how to configure the Kernel to use a VolatileMemoryStore with SemanticTextMemory and an OpenAI text embedding service.[37]

Java:

```
    Java



import com.azure.core.credential.AzureKeyCredential;
import com.azure.openai.OpenAIAsyncClient;
import com.azure.openai.OpenAIClientBuilder;
import com.microsoft.semantickernel.Kernel;
import
com.microsoft.semantickernel.aiservices.openai.chatcompletion.OpenAIChatCompleti
on;
import
com.microsoft.semantickernel.aiservices.openai.textembedding.OpenAITextEmbeddin
g;
import com.microsoft.semantickernel.memory.VolatileMemoryStore;

public class ExampleMemoryConnectors {
    public static void main(String args) {
        String apiKey = "your-api-key";
        String deploymentName = "your-deployment-name";
        String endpoint = "your-endpoint";
        String modelId = "your-model-id";
        String embeddingModelId = "text-embedding-ada-002";

        OpenAIAsyncClient client = new OpenAIClientBuilder()
            .credential(new AzureKeyCredential(apiKey))
            .endpoint(endpoint)
            .buildAsyncClient();

        OpenAIChatCompletion chatCompletionService =
OpenAIChatCompletion.builder()
            .withModelId(modelId)
            .withOpenAIAsyncClient(client)
            .build();

        OpenAITextEmbedding textEmbeddingGenerationService = new
OpenAITextEmbedding.Builder()
            .withModelId(embeddingModelId)
            .withOpenAIAsyncClient(client)
            .build();

        VolatileMemoryStore memoryStore = new VolatileMemoryStore();

        Kernel kernel = Kernel.builder()
            .withAIService(OpenAIChatCompletion.class, chatCompletionService)
            .withMemoryStore(VolatileMemoryStore.class, memoryStore)
            .withAIService(OpenAITextEmbedding.class,
textEmbeddingGenerationService)
            .build();
    }
}
```

This Java example illustrates configuring the Kernel to use VolatileMemoryStore along with OpenAI chat and text embedding services.

The availability of memory connectors for various vector databases [25] highlights the significance of semantic memory and Retrieval-Augmented Generation (RAG) in modern AI applications. These connectors empower developers to effortlessly integrate long-term memory capabilities into their AI agents, enabling them to access and reason over extensive amounts of information.

Memory

Memory in the context of AI agents built with Semantic Kernel refers to the ability of these agents to retain, retrieve, and utilize information from past interactions or experiences.[6] This capability allows agents to maintain context across conversations, recall important facts or user preferences, and ultimately exhibit more context-aware and adaptive behaviors.[36]

Semantic Kernel leverages memory to significantly enhance the capabilities of Large Language Models (LLMs). By providing LLMs with access to a broader range of information beyond their inherent training data, memory mechanisms enable them to generate more relevant and coherent responses.[35] This is particularly beneficial in addressing common limitations of LLMs, such as the tendency to generate inaccurate information, inconsistencies in factual details, or biases present in their training data.[35]

Semantic Kernel offers several distinct types of memory storage:

- **Volatile Memory:** This type of memory is temporary and short-lived, retaining data only for the duration of a single session or interaction.[36] It is particularly useful for managing immediate conversational context or task-related data that does not need to be persisted beyond the current session.[36] Volatile memory is often employed during prototyping or for one-shot interactions where long-term data retention is not required.[36]
- **Non-Volatile Memory:** In contrast to volatile memory, non-volatile memory is persistent, meaning that the data stored within it is retained over the long term and can be accessed across different sessions.[36] This type of memory is essential for maintaining user preferences, storing critical facts, or any other data that needs to be recalled in future interactions.[36]
- **Semantic Memory:** This refers to the structured storage of knowledge and concepts in a way that allows an AI agent to reference and utilize this information contextually.[36] Semantic memory often involves storing data as vector embeddings, which are numerical representations of the semantic meaning of text, enabling efficient similarity searches and the ability to draw insights and connections between different pieces of information.[36] Within the Semantic Kernel ecosystem, Semantic Memory is provided as a library within the SDK, acting as a wrapper around database interactions to simplify the storage and retrieval of text using embeddings.[36]
- **Kernel Memory:** представляє собою більш комплексну систему управління пам'яттю, яка може працювати у фоновому режимі як служба.[36] Вона надає REST API для полегшення створення рівня оркестрації RAG для агентних AI-рішень. Kernel Memory об'єднує кілька типів пам'яті (короткочасну, довготривалу тощо) для управління контекстом та станом у вашому додатку.[36] Вона підтримує різноманітні формати даних, крім тексту, такі як веб-сторінки, PDF-файли та зображення, і пропонує розширені функції, такі як підсумовування та OCR.[36]

Semantic Kernel achieves integration with various memory stores, including Azure Cognitive Search and vector databases like Qdrant, Chroma, Milvus, Pinecone, Redis, and SQL databases, through the use of memory connectors.[17] These connectors serve as bridges between the Kernel and specific memory store implementations, abstracting away the database-specific details and simplifying the process of storing, managing, and querying information.[41]

Here are code examples demonstrating how to save and retrieve information from different memory stores:

C#:

C#

```
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Memory;
using Microsoft.SemanticKernel.Connectors.Memory.Volatile;
using Microsoft.SemanticKernel.TextEmbedding;
using Microsoft.SemanticKernel.Connectors.OpenAI;

// Assuming a Kernel instance 'kernel' is initialized and OpenAI services are added

// Get the volatile memory store
var memoryStore = kernel.GetService<IMemoryStore>();

// Get the text embedding generation service
var embeddingGenerator = kernel.GetService<ITextEmbeddingGenerationService>();

// Create a semantic text memory instance
var semanticMemory = new SemanticTextMemory(memoryStore,
embeddingGenerator);

string collectionName = "MyCollection";
string documentId = "MyDocument";
string textToSave = "The capital of France is Paris.";

// Save information to memory
await semanticMemory.SaveInformationAsync(collectionName, documentId,
textToSave);

// Search for information in memory
MemoryQueryResult searchResult = await
semanticMemory.SearchAsync(collectionName, "What is the capital of France?", 1,
0.8).FirstOrDefaultAsync();

if (searchResult!= null)
{
    Console.WriteLine($"Found in memory: {searchResult.Metadata.Text}");
}
else
{
    Console.WriteLine("Information not found in memory.");
}
```

This C# example demonstrates saving a piece of information to volatile memory using semanticMemory.SaveInformationAsync and then searching for it using semanticMemory.SearchAsync.[46]

Python:

Python

```
import asyncio
from semantic_kernel import Kernel
from semantic_kernel.memory.volatile_memory_store import VolatileMemoryStore
from semantic_kernel.memory.semantic_text_memory import SemanticTextMemory
from semantic_kernel.connectors.ai.open_ai import OpenAITextEmbedding,
AzureChatCompletion

async def main():
    kernel = Kernel()
    # Add a chat service (replace with your actual configuration)
    kernel.add_service(AzureChatCompletion(service_id="azure_openai",
deployment_name="your-deployment-name", endpoint="your-endpoint",
api_key="your-api-key"))
```

```
    # Add OpenAI text embedding service
    kernel.add_service(OpenAITextEmbedding(service_id="openai_embedding", model_id="text-embedding-ada-002", api_key="your-openai-api-key"))

    # Use volatile memory store
    memory_store = VolatileMemoryStore()
    kernel.memory = SemanticTextMemory(storage=memory_store, embeddings_generator=kernel.get_service("openai_embedding"))

    collection_id = "aboutMe"

    # Add some documents to the semantic memory
    await kernel.memory.save_information(collection=collection_id, id="info1", text="My name is Andrea")
    await kernel.memory.save_information(collection=collection_id, id="info2", text="I currently work as a tour guide")

    # Search the populated memory store
    questions = [
        "what's my name",
        "what do I do for work"
    ]
    for question in questions:
        print(f"Question: {question}")
        result = await kernel.memory.search(collection_id, question)
        print(f"Answer: {result.text}\n")

if __name__ == "__main__":
    asyncio.run(main())
```

This Python example shows how to save information to volatile memory using kernel.memory.save_information and then search for it using kernel.memory.search.[37]

Table 1: Comparison of Semantic Memory and Kernel Memory

Feature	Kernel Memory	Semantic Memory

Data formats	Web pages, PDF, Images, Word, PowerPoint, Excel, Markdown, Text, JSON, HTML	Text only
Search	Cosine similarity, Hybrid search with filters (AND/OR conditions)	Cosine similarity
Language support	Any language, command line tools, browser extensions, etc.	C#, Python, Java
Storage engines	Azure AI Search, Elasticsearch, MongoDB Atlas, Qdrant, Redis, SQL Server, etc.	Azure AI Search, Chroma, DuckDB, Kusto, Milvus, etc.
File storage	Disk, Azure Blobs, AWS S3, MongoDB Atlas, In memory (volatile)	-
RAG support	Yes, with sources lookup	Yes, with additional libraries
Summarization	Yes	-
OCR	Yes via Azure Document Intelligence	-
Cloud deployment	Yes	Yes, but custom
LLM support	Azure OpenAI, OpenAI, Anthropic, Ollama, LLamaSharp, etc.	Azure OpenAI, OpenAI, Gemini, Hugging Face, etc.

This table provides a concise comparison of the key features of Semantic Memory and Kernel Memory, helping developers to understand their respective capabilities and choose the appropriate memory solution for their AI applications based on their specific requirements.[36]

The evolution from Semantic Memory to the more comprehensive Kernel Memory [36] indicates a trend towards increasingly versatile and feature-rich memory solutions for AI agents. The ability of Kernel Memory to handle diverse data types and offer advanced processing capabilities like summarization and OCR suggests a growing need for AI agents to process and remember information from a wide range of sources.

Planners

Planners are a crucial component of Semantic Kernel, acting as intelligent functions that take a user's request or goal and automatically generate a step-by-step plan to achieve it.[6] Planners leverage the power of AI to dynamically combine and orchestrate the various plugins registered within the Kernel, creating a sequence of actions (function calls) that will ultimately fulfill the user's objective.[16] This capability allows developers to create atomic, reusable functions that can be intelligently combined in unforeseen ways to address complex tasks.[49]

Semantic Kernel offers different types of planners, each with its own approach to generating and executing plans. Two key planners are the **Sequential Planner** and the **Stepwise Planner**.

Sequential Planner

The **Sequential Planner** is designed to create a plan consisting of multiple functions that are executed in a specific order, with the output of one function often serving as the input for the next.[6] This type of planner is particularly useful for scenarios where a task can be broken down into a linear sequence of steps, such as performing a web search, then summarizing the results, and finally sending the summary via email.[16]

Internally, the Sequential Planner utilizes an LLM to generate the plan, which is then parsed and executed as a series of Semantic Kernel functions.[52] The core functionality involves taking a user-provided goal as input and returning a plan that outlines the necessary steps to achieve it.[52] The planner ensures a smooth flow of data between the individual steps, passing the output of each function to the subsequent one as needed.[49]

Here are code examples demonstrating the use of the Sequential Planner:

C#:

C#

```
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Planners.Sequential;

// Assuming a Kernel instance 'kernel' with necessary plugins is initialized

// Instantiate a new Sequential Planner
SequentialPlanner planner = new SequentialPlanner(kernel);

// Define the goal
string goal = "Find the current weather in London and then send an email to myself with the weather
report.";

// Create a plan asynchronously based on the goal
Plan plan = await planner.CreatePlanAsync(goal);

// Execute the generated plan
KernelResult result = await kernel.RunAsync(plan);

Console.WriteLine($"Plan result: {result.GetValue<string>()}");
```

This C# example shows how to instantiate a SequentialPlanner, define a goal that involves multiple steps, create a plan using CreatePlanAsync, and then execute the plan using kernel.RunAsync.[51]

Python:

```
Python

import asyncio
from semantic_kernel import Kernel
from semantic_kernel.planners import SequentialPlanner
from semantic_kernel.connectors.ai.open_ai import AzureChatCompletion

async def main():
    kernel = Kernel()
    # Add a chat service (replace with your actual configuration)
    kernel.add_service(AzureChatCompletion(service_id="azure_openai",
deployment_name="your-deployment-name", endpoint="your-endpoint",
```

```python
api_key="your-api-key"))

    # Create a Sequential Planner
    planner = SequentialPlanner(kernel)

    # Define the task
    ask = """Tomorrow is Valentine's day. I need to come up with a few short poems.
   She likes Shakespeare so write using his style. She speaks French so write it in French.
   Convert the text to uppercase.
   """;

    # Create a plan
    sequential_plan = await planner.create_plan(goal=ask)

    # Execute the plan
    result = await sequential_plan.invoke(kernel)

    print(result)

if __name__ == "__main__":
    asyncio.run(main())
```

This Python example demonstrates creating a SequentialPlanner, defining a complex task involving multiple transformations, creating a plan using planner.create_plan, and then executing it.[18]

Java:

Java

```java
import com.microsoft.semantickernel.Kernel;
import com.microsoft.semantickernel.planners.SequentialPlanner;
import com.microsoft.semantickernel.plugin.KernelPlugin;
import com.microsoft.semantickernel.plugin.functional.KernelPluginFactory;

public class ExampleSequentialPlanner {
    public static void main(String args) {
        Kernel kernel = Kernel.builder().build();

        // Import necessary plugins (example: WriterPlugin and SummarizePlugin)
        kernel.importSkillFromDirectory("WriterSkill", "src/main/resources/Skills", "WriterSkill");
        kernel.importSkillFromDirectory("SummarizeSkill", "src/main/resources/Skills",
"SummarizeSkill");

        // Create a Sequential Planner
        SequentialPlanner planner = new SequentialPlanner(kernel, null, null);

        // Define the text to summarize and translate
        String textToSummarize = "The quick brown fox jumps over the lazy dog. This is a simple
sentence.";
        String goal = textToSummarize + " ===== summarize the above content and then translate
it to Dutch. =====";

        // Create a plan
        Mono<Plan> result = planner.createPlanAsync(goal);

        // Execute the plan (implementation details for execution depend on the Java SDK)
        //...
    }
}
```

This Java example illustrates creating a SequentialPlanner, importing necessary plugins, defining a goal that involves summarization and translation, and creating a plan.[54]

The Sequential Planner simplifies the development of multi-step automated workflows by allowing the AI to determine the necessary sequence of function calls. This reduces the need for developers to manually orchestrate each step in the process.

Stepwise Planner

The **Stepwise Planner** takes a different approach to task automation by breaking down complex tasks into smaller, more manageable steps.[16] It operates based on a neuro-symbolic architecture known as MRKL (Modular Reasoning, Knowledge and Language), allowing for a structured and iterative problem-solving process.[18] This planner is particularly well-suited for scenarios that require dynamic selection of plugins and the handling of intricate requests with interconnected steps.[57]

The Stepwise Planner works by generating a "thought" process, evaluating the available pathways to achieve the goal, then performing an "action" (typically invoking a function), evaluating the response, and potentially iterating through this cycle until a "final_answer" is produced.[59] This dynamic and adaptive nature allows the planner to learn from the outcomes of each step and adjust its subsequent actions accordingly.[57]

Here are code examples demonstrating the use of the Stepwise Planner:

C#:

```csharp
C#

using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Planners.Stepwise;

// Assuming a Kernel instance 'kernel' with necessary plugins is initialized

// Create a Stepwise Planner
var planner = new FunctionCallingStepwisePlanner();

// Define a complex question
string question = "What is the capital of France? Who is that city's current mayor? What percentage of their life has been in the 21st century as of today?";

// Execute the plan
FunctionCallingStepwisePlannerResult result = await planner.ExecuteAsync(kernel, question);

Console.WriteLine($"Final Answer: {result.FinalAnswer}");
```

This C# example shows how to create a FunctionCallingStepwisePlanner and execute it with a complex, multi-part question.[59]

Python:

Python

```python
import asyncio
from semantic_kernel import Kernel
from semantic_kernel.planners.stepwise_planner import StepwisePlanner
from semantic_kernel.connectors.ai.open_ai import AzureChatCompletion

async def main():
    kernel = Kernel()
    # Add a chat service (replace with your actual configuration)
    kernel.add_service(AzureChatCompletion(service_id="azure_openai",
deployment_name="your-deployment-name", endpoint="your-endpoint",
api_key="your-api-key"))

    # Create a Stepwise Planner
    planner = StepwisePlanner(kernel=kernel)

    # Define the goal
    ask = "Write a business mail to share the population of the United States in 2015. Make sure to
specify how many people, among the population, identify themselves as male and female. Don't share
approximations, please share the exact numbers.";

    # Create a plan
    plan = await planner.create_plan(goal=ask)

    # Execute the plan
    result = await plan.invoke(kernel=kernel)

    print(result)

if __name__ == "__main__":
    asyncio.run(main())
```

This Python example demonstrates creating a StepwisePlanner and executing it with a goal that requires multiple steps and precise information retrieval.[18]

Java:

Java

```java
import com.microsoft.semantickernel.Kernel;
import com.microsoft.semantickernel.planners.StepwisePlanner;

public class ExampleStepwisePlanner {
    public static void main(String args) {
        Kernel kernel = Kernel.builder().build();

        // Import necessary plugins (example: UnitedStatesPlugin and MailPlugin)
        kernel.importSkillFromDirectory("UnitedStatesPlugin", "src/main/resources/Plugins",
"UnitedStatesPlugin");
        kernel.importSkillFromDirectory("MailPlugin", "src/main/resources/Plugins", "MailPlugin");

        // Create a Stepwise Planner
        StepwisePlanner planner = new StepwisePlanner(kernel, null);

        // Define the ask
        String ask = "Write a mail to share the number of the United States population in 2015 for a
research program.";

        // Create a plan
        Mono<Plan> originalPlan = planner.createPlanAsync(ask);

        // Execute the plan (implementation details for execution depend on the Java SDK)
        //...
    }
}
```

This Java example illustrates creating a StepwisePlanner, importing necessary plugins, defining an ask, and creating a plan.[54]

The Stepwise Planner offers a more advanced mechanism for planning and executing complex tasks compared to the Sequential Planner. Its iterative "thought" and "action" cycle enables AI agents to tackle intricate problems that may require reasoning and multiple steps to reach a solution.

3. Beginner's Guide to Using Semantic Kernel

For newcomers to Semantic Kernel, the official documentation provides excellent introductory guides and beginner-level tutorials in the "Getting Started" and "Quick Start" sections.[5] These resources are specifically designed to help new users understand the fundamental concepts and begin working with the framework.

The "Quick Start Guide" [9] offers step-by-step instructions on how to:

- Install the necessary SDK packages for your preferred programming language (.NET, Python, or Java).
- Create a basic AI agent capable of engaging in a back-and-forth conversation with a user.
- Give an AI agent the ability to execute your custom code through the creation and use of plugins.
- Observe the AI dynamically generate execution plans using automatic function calling to achieve user requests.

The guide provides illustrative code examples in all three supported languages, focusing on core tasks such as:

- **Initializing the Kernel:** Creating an instance of the central Kernel object, which acts as the orchestrator for all AI operations.
- **Adding an AI Service:** Configuring and adding an AI service, such as Azure OpenAI, to the Kernel builder by providing the necessary API credentials and endpoint details.
- **Creating and Registering a Simple Plugin:** Developing either a basic semantic function (defined directly within the code or loaded from a prompt template file) or a simple native function (implemented in code and annotated accordingly) and registering it with the Kernel using methods like AddPlugin or ImportFunctions.
- **Invoking a Function:** Instructing the Kernel to execute a function from a registered plugin, either by using methods like InvokePromptAsync for semantic functions or through the chat completion service with function calling enabled for both semantic and native functions.

The availability of these readily accessible resources, including step-by-step guides and practical code examples, demonstrates a clear focus on making Semantic Kernel easy to adopt for developers who are new to the framework. This approach allows beginners to quickly grasp the foundational concepts and start experimenting with the SDK's capabilities to build their own AI-powered applications.

4. Intermediate Level Concepts and Implementation

Creating Custom Semantic and Native Plugins

At the intermediate level, developers can delve into creating their own custom plugins to extend the capabilities of Semantic Kernel beyond the built-in functionalities.

Semantic Plugins (Intermediate)

Creating custom semantic plugins involves defining a folder structure that typically includes a skprompt.txt file containing the natural language prompt template and a config.json file that holds the configuration details for the prompt, such as its description, input variables, and execution settings.[14] The prompt template itself can utilize a templating language (like Handlebars) to include expressions and variables, allowing for dynamic prompt generation based on input parameters.[14]

Use Cases and Examples:

- A WriterPlugin could be created with a semantic function that takes a piece of text as input and rewrites it in the style of a famous author, like Shakespeare.[18] The prompt template would include instructions on adopting Shakespearean language, and the config.json would define the input variable for the text to be rewritten.
- A travel application might include a SuggestDestinationsPlugin with a semantic function that takes user preferences (e.g., interests, budget) as input and suggests suitable travel destinations.[14] The prompt would be crafted to ask an LLM for destination recommendations based on these preferences.
- A semantic function for translating text between languages could be created. The prompt would instruct the LLM to translate the input text (passed via ContextVariables) to a specified target language.[6]

Code Snippets (Illustrative):

C# (Loading Semantic Functions from a Directory):

C#

```
using Microsoft.SemanticKernel;

// Assuming a Kernel instance 'kernel' is initialized
```

```
string pluginsDirectory = Path.Combine(Directory.GetCurrentDirectory(), "Plugins");
kernel.ImportPluginFromPromptDirectory(pluginsDirectory, "MyCustomPlugin");

// Now you can invoke functions from the "MyCustomPlugin"
KernelResult result = await kernel.InvokeAsync("MyCustomPlugin", "MySemanticFunction",
new KernelArguments() { ["input"] = "Some text to process" });
Console.WriteLine(result.GetValue<string>());
```

This C# example shows how to load semantic functions from a directory named "Plugins" into a plugin named "MyCustomPlugin".[22]

Python (Defining a Semantic Function):

Python

```python
from semantic_kernel import Kernel
from semantic_kernel.functions import KernelFunctionFromPrompt
from semantic_kernel.connectors.ai.open_ai import
OpenAIChatPromptExecutionSettings

async def main():
    kernel = Kernel()
    # Add your AI service

    prompt = """{{$input}}
   Summarize the above text in three sentences.
   """;

    execution_settings = OpenAIChatPromptExecutionSettings(temperature=0.5,
max_tokens=150)

    summarize_function = KernelFunctionFromPrompt(
        prompt=prompt,
        execution_settings=execution_settings,
        name="Summarize",
        description="Summarizes a given text."
    )
    kernel.add_function(plugin_name="TextProcessing", function=summarize_function)

    result = await kernel.invoke("TextProcessing", "Summarize", input="A long piece of text needs to be summarized...")
    print(result)

if __name__ == "__main__":
    asyncio.run(main())
```

This Python example defines a semantic function named "Summarize" that takes text as input and summarizes it.[18]

Native Plugins (Intermediate)

Developing native plugins involves creating classes in C#, Python, or Java and annotating their methods with the appropriate Kernel function decorators or attributes.[20] The [Description] attribute (C#) or description parameter (@kernel_function in Python) is crucial for providing semantic meaning to the functions and their parameters, enabling the AI agent to understand their purpose and how to use them effectively.[20]

Use Cases and Examples:

- A DailyFactPlugin in C# could contain a native function that fetches an interesting historical fact for the current date from an external API or a local data source.[19]
- A BookingPlugin in C# might have a native function that checks the availability of appointments or makes a new booking, interacting with a calendar or scheduling system.[21]
- A WeatherPlugin in Python could include a native function that retrieves the current weather conditions for a given city using a weather API.[24]
- A MathPlugin (often built-in or custom) could provide native functions for performing various mathematical operations like addition, subtraction, multiplication, and division.[13]

Code Snippets (Illustrative):

C# (Creating a Native Plugin):

```
C#
```

```
using Microsoft.SemanticKernel;
using System.ComponentModel;

public class CalculatorPlugin
{
    [KernelFunction, Description("Adds two numbers.")]
    public int Add(
        int num1,
        int num2)
    {
        return num1 + num2;
    }


    public int Subtract(
        int num1,
        int num2)
    {
        return num1 - num2;
    }
}

// In your main application:
Kernel kernel = Kernel.CreateBuilder().Build();
kernel.ImportPluginFromObject(new CalculatorPlugin(), "Calculator");
```

This C# example defines a CalculatorPlugin with native functions for addition and subtraction, both clearly described for the AI.[19]

Python (Creating a Native Plugin):

Python

```
from semantic_kernel import kernel_function, Kernel
import os

class FileIOPlugin:
  @kernel_function(description="Read text from a file")
  def read_file(self, file_path: str) -> str:
      """Reads all text from a file."""
      with open(file_path, 'r') as f:
          return f.read()

  @kernel_function(description="Write text to a file")
  def write_file(self, file_path: str, text: str) -> None:
      """Writes text to a file."""
      with open(file_path, 'w') as f:
          f.write(text)

async def main():
  kernel = Kernel()
  # Add your AI service

  file_io_plugin = FileIOPlugin()
  kernel.add_plugin(file_io_plugin, plugin_name="FileIO")

if __name__ == "__main__":
  asyncio.run(main())
```

This Python example creates a FileIOPlugin with native functions to read and write text to files.[18]

Integrating and Utilizing Different Memory Stores

At the intermediate level, developers learn how to configure and utilize various memory connectors to integrate different memory stores into their Semantic Kernel applications. This enables the creation of AI agents with the ability to persist and retrieve information, crucial for building context-aware and long-lasting applications.

Memory Store Integration:

- **Azure AI Search:** Semantic Kernel can leverage Azure AI Search (formerly Azure Cognitive Search) as a robust and scalable memory store. This involves configuring the appropriate connector with your Azure AI Search service endpoint and API key. Once connected, developers can use the Semantic Kernel's memory APIs to index and search documents and other data stored in Azure AI Search.[25]
- **Vector Databases (Qdrant, Chroma, Milvus, Pinecone, Redis, etc.):** Semantic Kernel provides connectors for a variety of popular vector databases. These

databases are optimized for storing and querying high-dimensional vector embeddings, making them ideal for semantic search and Retrieval-Augmented Generation (RAG) scenarios. Integration typically involves installing the specific connector package and configuring it with the database's connection details.[25]

- **Volatile Memory:** For simpler use cases or testing, Semantic Kernel offers a built-in volatile memory store that resides in the application's memory. This requires minimal configuration and is useful for temporary storage of information during a session.[36]

Code Snippets (Illustrative):

C# (Using Pinecone Memory Store):

C#

```csharp
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Memory;
using Microsoft.SemanticKernel.Connectors.Memory.Pinecone;
using Microsoft.SemanticKernel.TextEmbedding;
using Microsoft.SemanticKernel.Connectors.OpenAI;

// Assuming a Kernel builder 'builder' and OpenAI configuration

// Add OpenAI text embedding generation service
builder.Services.AddSingleton<ITextEmbeddingGenerationService>(new
OpenAITextEmbeddingGenerationService(
    modelId: "text-embedding-ada-002",
    apiKey: "your-openai-api-key"));

// Add Pinecone memory store
builder.Services.AddSingleton<IMemoryStore>(new PineconeMemoryStore(
    apiKey: "your-pinecone-api-key",
    environment: "your-pinecone-environment"));

// Build the Kernel
Kernel kernel = builder.Build();

// Use the memory store to save and search information
```

```
var memoryStore = kernel.GetService<IMemoryStore>();
string collectionName = "MyPineconeCollection";
string documentId = "Document1";
string text = "Information to store in Pinecone.";
await memoryStore.UpsertAsync(collectionName, new MemoryRecord(documentId,
text, null, null, null));
var results = await memoryStore.SearchAsync(collectionName, "Information to find", 1,
0.8).ToListAsync();
```

This C# example demonstrates configuring and using the Pinecone memory store within a Semantic Kernel application.[46]

Python (Using Astra DB Serverless as Memory Store):

Python

```python
import asyncio
from semantic_kernel import Kernel
from semantic_kernel.connectors.ai.open_ai import OpenAITextEmbedding
from semantic_kernel.connectors.memory.astradb import AstraDBMemoryStore

async def main():
    kernel = Kernel()
    # Add your AI service

    # Configure Astra DB Serverless
    astra_db_store = AstraDBMemoryStore(
        token="your-astra-db-token",
        api_endpoint="your-astra-db-api-endpoint",
        embedding_dimension=1536,  # Example dimension
    )
    kernel.memory = SemanticTextMemory(storage=astra_db_store,
embeddings_generator=kernel.get_service(OpenAITextEmbedding))

    collection_id = "my_astra_collection"
    await kernel.memory.save_information(collection=collection_id, id="doc1", text="Some
information")
```

```
    results = await kernel.memory.search(collection=collection_id, query="information", limit=1)
    print(results.text)

if __name__ == "__main__":
    asyncio.run(main())
```

This Python example shows how to set up and use Astra DB Serverless as the memory store for Semantic Kernel.[39]

By integrating these different memory stores, developers can equip their AI agents with various capabilities, from simple in-memory caching to persistent semantic search over large datasets, enabling more sophisticated and context-aware AI applications.

Practical Examples of Using Sequential and Stepwise Planners

At the intermediate level, developers can apply their understanding of Sequential and Stepwise Planners to automate more complex workflows.

Sequential Planner (Intermediate Examples)

- **Fetching, Processing, and Reporting Data:** A Sequential Planner could be used to automate the process of fetching data from multiple sources (e.g., a database and a web API), then processing this data (e.g., performing calculations or transformations), and finally generating a report or sending an email with the processed information.[52] Each step in the plan would correspond to a specific plugin function responsible for data retrieval, processing, or reporting.
- **Content Creation Workflow:** Another use case involves a Sequential Planner that first uses a semantic function to brainstorm content ideas based on a user's topic, then another semantic function to generate a draft of the content, and finally a native function to save the draft to a file or publish it to a platform.[18]

Code Snippet (Illustrative - C#):

C#

```
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Planners.Sequential;
```

```
// Assuming a Kernel instance 'kernel' with plugins for web search, summarization, and email

SequentialPlanner planner = new SequentialPlanner(kernel);

string goal = "Search for the latest news on AI, summarize the top 3 articles, and email the summary to my address.";

Plan plan = await planner.CreatePlanAsync(goal);

KernelResult result = await kernel.RunAsync(plan);

Console.WriteLine($"Plan result: {result.GetValue<string>()}");
```

This C# example outlines a sequential plan to search for news, summarize it, and send an email, assuming the existence of appropriate plugins.[18]

Stepwise Planner (Intermediate Examples)

- **Multi-Part Question Answering:** The Stepwise Planner excels at handling intricate questions that require reasoning and the use of multiple tools. For example, answering a question that requires fetching information from a website, performing a calculation on the retrieved data, and then providing a final answer.[59] Each step in the plan would involve a "thought" about what needs to be done next and an "action" to execute a relevant plugin function.
- **Task Completion with Dependencies:** The Stepwise Planner can manage tasks that have dependencies. For instance, if a user asks to "book a flight and then add the flight details to my calendar," the planner could first invoke a flight booking plugin, extract the flight details from the response, and then invoke a calendar plugin to add an event with those details.[18]

Code Snippet (Illustrative - Python):

Python

```python
import asyncio
from semantic_kernel import Kernel
from semantic_kernel.planners.stepwise_planner import StepwisePlanner
from semantic_kernel.connectors.ai.open_ai import AzureChatCompletion

async def main():
    kernel = Kernel()
    # Add your AI service and necessary plugins (e.g., for web search and calculation)

    planner = StepwisePlanner(kernel=kernel)

    ask = "What is the current price of Microsoft stock, and what was the average price last month?";

    plan = await planner.create_plan(goal=ask)

    result = await plan.invoke(kernel=kernel)

    print(result)

if __name__ == "__main__":
    asyncio.run(main())
```

This Python example illustrates using a Stepwise Planner to answer a question that likely requires multiple steps, such as fetching the current stock price and then calculating the average price for the previous month.[18]

These examples demonstrate how intermediate-level developers can leverage the power of Sequential and Stepwise Planners to automate increasingly complex and intelligent workflows within their Semantic Kernel applications.

5. Advanced Topics in Semantic Kernel

As developers gain more experience with Semantic Kernel, they can explore advanced topics to further enhance the capabilities and robustness of their AI applications.

Developing Custom Connectors (Advanced)

In scenarios where developers need to integrate Semantic Kernel with specialized or proprietary AI models or services that are not supported by the SDK's built-in connectors, the ability to create **custom connectors** becomes essential.[26] While the official documentation might not provide a step-by-step guide for building custom connectors, the framework's design, particularly the presence of abstractions for various AI service types, indicates that extensibility is a key consideration.[30]

The core of this extensibility lies in the underlying interfaces and abstract classes within Semantic Kernel that define the contract for interacting with different AI service functionalities. For example, interfaces like IChatCompletionService, ITextGenerationService, and ITextEmbeddingGenerationService (in.NET) or their equivalents in Python and Java, outline the methods and properties that a connector for a specific type of AI service must implement.[30]

The high-level process for building a custom connector typically involves the following steps:

1. **Analyze the Target AI Service:** Developers need to thoroughly understand the API endpoints, request and response formats, authentication methods, and any specific requirements of the AI service they intend to connect to.[26]
2. **Implement the Relevant Semantic Kernel Interface:** A new class needs to be created that implements the appropriate Semantic Kernel service interface (e.g., IChatCompletionService for a chat model). This class will act as the custom connector.
3. **Implement Interface Methods:** Within the custom connector class, developers must implement the methods defined by the Semantic Kernel interface. This typically involves making API calls to the target AI service, handling the formatting of requests according to the service's specifications, and parsing the responses received from the service into a format that Semantic Kernel can understand. This might also include handling authentication and error scenarios specific to the target AI service.
4. **Register the Custom Connector with the Kernel:** Once the custom connector is implemented, it needs to be registered with the Kernel's service provider. This makes the custom connector available for use within the Semantic Kernel application, allowing it to be invoked by plugins and planners as needed.

The existence of these base interfaces for different AI functionalities [30] implies that developers can create custom implementations to connect to virtually any AI service. This extensibility is a powerful feature for advanced users who have unique integration requirements or want to leverage AI models or services that are not yet directly supported by the Semantic Kernel SDK.

Advanced Plugin Development Techniques (Advanced)

Semantic Kernel offers several advanced techniques for developing more sophisticated and efficient plugins [12]:

- **Transforming Semantic Kernel Functions:** This technique allows developers to modify the behavior, context, parameters, and return types of existing Semantic

Kernel functions without altering their original implementations.[12] This can be useful in scenarios where the default behavior of a function does not align with the desired outcome, or when additional context information needs to be provided to a function that the LLM cannot infer on its own. Function transformation can also simplify complex function signatures or adapt them to better suit the LLM's understanding.
- **Local State Utilization:** For plugins that handle large or confidential datasets, such as documents or emails containing sensitive information, utilizing local state can be a more efficient and secure approach.[12] Instead of passing the entire dataset to the LLM, the plugin can store the data locally and pass only a state ID. Functions within the plugin can then accept and return this state ID, allowing the application to look up and access the data locally. This method can significantly reduce token consumption and enhance data privacy by minimizing the amount of sensitive information sent to the LLM.
- **Providing Function Return Type Schema to AI Model:** To improve the accuracy and reliability of function calls, developers can employ techniques to explicitly provide the schema of a function's return type to the AI model.[12] This helps the AI model to accurately identify the intended properties within the function's response, reducing potential inaccuracies that might arise if the model makes assumptions based on incomplete or ambiguous information. A well-defined return type schema leads to more precise and predictable outcomes from function calls.

Enterprise-Level Considerations (Advanced)

For developers building Semantic Kernel applications for enterprise environments, several advanced considerations become paramount:

- **Observability:** In enterprise-grade AI solutions, **observability** is crucial for monitoring and analyzing the internal state of the application.[5] Semantic Kernel addresses this by emitting telemetry data in the form of logs, metrics, and traces, all of which are compatible with the OpenTelemetry standard.[68] This allows developers to gain insights into the behavior and performance of their AI services, track the flow of execution, identify potential issues or bottlenecks, and measure key performance indicators like function invocation duration and token usage.[68] Integrating with monitoring and analysis tools like Azure Monitor and Application Insights can provide valuable dashboards and alerts for maintaining the health and efficiency of Semantic Kernel applications.[68]
- **Security:** Building secure AI applications is of utmost importance in enterprise settings.[5] Semantic Kernel provides a framework for integrating security best

practices, including secure authentication and authorization mechanisms, such as leveraging Microsoft Graph with appropriate permissions when interacting with Microsoft services.[76] Preventing prompt injection attacks, where malicious users attempt to manipulate the AI's behavior through crafted prompts, is another critical security consideration. This can be mitigated through careful input validation and potentially by using filters to sanitize or control the content of prompts.[69] Additionally, when dealing with multi-tenant solutions, ensuring proper data segregation and access control is essential to prevent data leakage between different tenants.[74]
- **Filters: Filters** in Semantic Kernel offer a powerful mechanism to intercept and modify the execution of functions and the rendering of prompts, enabling developers to implement security measures, responsible AI practices, and custom logic within their AI-powered applications.[5] There are three main types of filters: **Function Invocation Filters**, which run every time a KernelFunction is called, allowing for actions like accessing function arguments, handling exceptions, overriding results for caching or responsible AI scenarios, and implementing retry logic.[80] **Prompt Render Filters** are activated before the prompt rendering operation, enabling developers to view and modify the prompt before it is sent to the AI, for example, for PII redaction or semantic caching.[80] **Auto Function Invocation Filters** operate within the scope of automatic function calling, providing additional context about the chat history and allowing for control over the sequence and termination of function calls.[80] By strategically using filters, developers can enhance the security, reliability, and ethical considerations of their Semantic Kernel applications.

Deployment Considerations (Advanced)

Deploying Semantic Kernel applications in production environments requires careful consideration of several factors to ensure scalability, reliability, and maintainability:

- **Scalability:** As the usage of AI applications grows, the ability to **scale** the underlying infrastructure becomes crucial.[71] Semantic Kernel applications can be designed to distribute processing across multiple nodes or agents, often leveraging cloud computing platforms like Microsoft Azure, Amazon Web Services (AWS), or Google Cloud Platform (GCP) for dynamic resource allocation based on demand.[72] Containerization technologies like Docker and orchestration platforms like Kubernetes can also play a significant role in managing and scaling Semantic Kernel deployments.[83]
- **Infrastructure:** The choice of **infrastructure** for deploying Semantic Kernel applications depends on various factors, including the scale of the application,

performance requirements, budget constraints, and existing IT infrastructure.[72] Options range from fully cloud-based deployments using services like Azure Functions (serverless) or virtual machines on cloud providers, to on-premise deployments using container orchestration or traditional server infrastructure.[72] Hybrid cloud strategies, where applications are deployed across a combination of on-premise and cloud resources, can also be considered.[83]

- **Monitoring:** Once a Semantic Kernel application is deployed, continuous **monitoring** of its health and performance is essential.[69] This involves tracking key metrics such as request latency, response times, error rates, and resource utilization (CPU, memory, network). Utilizing monitoring tools provided by cloud platforms (e.g., Azure Monitor, AWS CloudWatch, Google Cloud Monitoring) or third-party solutions can help identify potential issues, performance bottlenecks, and areas for optimization. Setting up alerts based on predefined thresholds can enable proactive intervention to ensure the application remains healthy and responsive.[69]

6. Conclusion and Further Resources

Microsoft Semantic Kernel offers a powerful and flexible framework for developers looking to integrate the latest advancements in AI Large Language Models into their applications. By providing a comprehensive set of tools for orchestration, integration, memory management, and planning, Semantic Kernel simplifies the complexities of building intelligent, AI-powered solutions. Its model-agnostic design and support for multiple programming languages make it accessible to a wide range of developers, while its advanced features cater to the needs of enterprise-level applications.

Developers should consider adopting Semantic Kernel when there is a need to combine the natural language understanding and generation capabilities of LLMs with custom code and integrations to various AI services and memory stores. The framework's modular architecture and robust planning capabilities enable the creation of sophisticated AI agents capable of automating complex tasks and providing rich, context-aware interactions.

To continue learning and exploring the capabilities of Semantic Kernel, developers are encouraged to refer to the following resources:

- **Official Microsoft Semantic Kernel Documentation:** https://learn.microsoft.com/en-us/semantic-kernel/ [5]
- **Semantic Kernel GitHub Repository:** https://github.com/microsoft/semantic-kernel [2]
- **Semantic Kernel Blog:** https://devblogs.microsoft.com/semantic-kernel/ [5]

- **Semantic Kernel Discord Community:** https://github.com/microsoft/semantic-kernel/discussions [5]
- **NuGet Packages for.NET:**(https://www.nuget.org/packages?q=Microsoft.SemanticKernel) [86]
- **PyPI Package for Python:** https://pypi.org/project/semantic-kernel/
- **Detailed Samples:** https://learn.microsoft.com/en-us/semantic-kernel/get-started/detailed-samples [87]

By leveraging these resources and continuing to explore the features and functionalities of Semantic Kernel, developers can unlock the full potential of AI and build innovative applications that push the boundaries of what's possible.

Works cited

1. Semantic Kernel | AI Hub - Azure documentation, accessed on April 10, 2025, https://azure.github.io/aihub/docs/concepts/semantic-kernel/
2. MicrosoftDocs/semantic-kernel-docs - GitHub, accessed on April 10, 2025, https://github.com/MicrosoftDocs/semantic-kernel-docs
3. microsoft/semantic-kernel: Integrate cutting-edge LLM technology quickly and easily into your apps - GitHub, accessed on April 10, 2025, https://github.com/microsoft/semantic-kernel/
4. Introduction to Semantic Kernel | Microsoft Learn, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/overview/index
5. Semantic Kernel documentation | Microsoft Learn, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/
6. Semantic Kernel: The New Way to Create Artificial Intelligence Applications - Medium, accessed on April 10, 2025, https://medium.com/globant/semantic-kernel-the-new-way-to-create-artificial-intelligence-applications-7959d5fc90ca
7. microsoft/semantic-kernel: Integrate cutting-edge LLM technology quickly and easily into your apps - GitHub, accessed on April 10, 2025, https://github.com/microsoft/semantic-kernel
8. Building AI agents with the Semantic Kernel SDK and Azure OpenAI | Will Velida, accessed on April 10, 2025, https://www.willvelida.com/posts/intro-to-semantic-kernel/
9. How to quickly start with Semantic Kernel | Microsoft Learn, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/get-started/quick-start-guide
10. Updated Documentation for Semantic Kernel - Microsoft Developer Blogs, accessed on April 10, 2025, https://devblogs.microsoft.com/semantic-kernel/updated-documentation-for-semantic-kernel/
11. Understanding the kernel in Semantic Kernel | Microsoft Learn, accessed on April

10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/concepts/kernel
12. Plugins in Semantic Kernel | Microsoft Learn, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/concepts/plugins/
13. Creating Plugins with the Semantic Kernel SDK and C# | Will Velida, accessed on April 10, 2025, https://www.willvelida.com/posts/create-plugins-semantic-kernel/
14. Create Plugins for Semantic Kernel | by Nicolas Anderson | Medium, accessed on April 10, 2025, https://medium.com/@nicolas-31/create-plugins-for-semantic-kernel-fdd9e3f24d2e
15. Creating Plugins with the Semantic Kernel SDK and C# - DEV Community, accessed on April 10, 2025, https://dev.to/willvelida/creating-plugins-for-semantic-kernel-sdk-1ai4
16. Diving into Microsoft Semantic Kernel | by Prarthana Saikia - Medium, accessed on April 10, 2025, https://medium.com/@prarthana1/diving-into-microsoft-semantic-kernel-82e037fe427c
17. Semantic Kernel Microsoft Documentation | Restackio, accessed on April 10, 2025, https://www.restack.io/p/semantic-kernel-answer-microsoft-documentation-cat-ai
18. semantic-kernel/python/samples/getting_started/05-using-the-planner.ipynb at main, accessed on April 10, 2025, https://github.com/microsoft/semantic-kernel/blob/main/python/samples/getting_started/05-using-the-planner.ipynb
19. Semantic Kernel Hello World Plugins Part 2 - Jason Haley, accessed on April 10, 2025, https://jasonhaley.com/2024/04/26/semantic-kernel-hello-world-plugin-part2/
20. Add native code as a plugin - Learn Microsoft, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/concepts/plugins/adding-native-plugins
21. Semantic Kernel: Implementing Native Functions and Plugins - Jamie Maguire, accessed on April 10, 2025, https://jamiemaguire.net/index.php/2024/07/05/semantic-kernel-implementing-native-functions-and-plugins/
22. Semantic Kernel - Native plugins - The Developer's Cantina, accessed on April 10, 2025, https://www.developerscantina.com/p/semantic-kernel-native-plugins/
23. Plugin execution using semantic kernel causes error #9210 - GitHub, accessed on April 10, 2025, https://github.com/microsoft/semantic-kernel/discussions/9210
24. semantic-kernel/python/samples/concepts/plugins/openai_function_calling_with_custom_plugin.py at main - GitHub, accessed on April 10, 2025, https://github.com/microsoft/semantic-kernel/blob/main/python/samples/concepts/plugins/openai_function_calling_with_custom_plugin.py
25. Semantic Kernel Connectors Overview | Restackio, accessed on April 10, 2025, https://www.restack.io/p/semantic-kernel-answer-connectors-cat-ai
26. Chatbot with Semantic Kernel - Part 6: AI Connectors - DEV Community,

accessed on April 10, 2025, https://dev.to/davidgsola/chatbot-with-semantic-kernel-part-6-ai-connectors-4b63
27. How to Use SemanticKernel with OpenAI and Azure OpenAI in C#, accessed on April 10, 2025, https://techcommunity.microsoft.com/blog/educatordeveloperblog/how-to-use-semantickernel-with-openai-and-azure-openai-in-c/4081648
28. Package Microsoft.SemanticKernel.Connectors.OpenAI - GitHub, accessed on April 10, 2025, https://github.com/orgs/microsoft/packages/nuget/package/Microsoft.SemanticKernel.Connectors.OpenAI
29. Building your AI Agents using Python, HuggingFace Models and Semantic Kernel - Medium, accessed on April 10, 2025, https://medium.com/@nimritakoul01/building-your-ai-agents-using-python-huggingface-models-and-semantic-kernel-a2242432da30
30. Add chat completion services to Semantic Kernel | Microsoft Learn, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/concepts/ai-services/chat-completion/
31. Loading a Huggingface Model with Microsofts Semantic Kernel in C# / VB.NET, accessed on April 10, 2025, https://stackoverflow.com/questions/77110608/loading-a-huggingface-model-with-microsofts-semantic-kernel-in-c-sharp-vb-net
32. Semantic Kernel Components | Microsoft Learn, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/concepts/semantic-kernel-components
33. Say hello to the updated Semantic Kernel docs!, accessed on April 10, 2025, https://devblogs.microsoft.com/semantic-kernel/say-hello-to-the-updated-semantic-kernel-docs/
34. semantic-kernel/docs/decisions/0050-updated-vector-store-design.md at main - GitHub, accessed on April 10, 2025, https://github.com/microsoft/semantic-kernel/blob/main/docs/decisions/0050-updated-vector-store-design.md
35. What are Semantic Kernel Vector Store connectors? (Preview) - Learn Microsoft, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/concepts/vector-store-connectors/
36. Semantic Kernel: Using Memories to Create Intelligent AI Agents - Jamie Maguire, accessed on April 10, 2025, https://jamiemaguire.net/index.php/2024/10/06/semantic-kernel-using-memories-to-create-intelligent-ai-agents/
37. semantic-kernel/python/samples/getting_started/06-memory-and-embeddings.ipynb at main, accessed on April 10, 2025, https://github.com/microsoft/semantic-kernel/blob/main/python/samples/getting_started/06-memory-and-embeddings.ipynb

38. Legacy Semantic Kernel Memory Stores - Learn Microsoft, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/concepts/vector-store-connectors/memory-stores
39. Integrate Semantic Kernel with Astra DB Serverless - DataStax Docs, accessed on April 10, 2025, https://docs.datastax.com/en/astra-db-serverless/integrations/semantic-kernel.html
40. Semantic Kernel: Using Memories to Create Intelligent AI Agents - Azure Feeds, accessed on April 10, 2025, https://azurefeeds.com/2024/10/07/semantic-kernel-using-memories-to-create-intelligent-ai-agents/
41. What are Semantic Kernel Vector Store connectors? (Preview ..., accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/concepts/vector-store-connectors/index
42. Build a custom Copilot experience with your private data using and Kernel Memory, accessed on April 10, 2025, https://www.developerscantina.com/p/kernel-memory/
43. 5. Using Embeddings and Semantic Memory in Semantic Kernel Apps - YouTube, accessed on April 10, 2025, https://www.youtube.com/watch?v=FRf-7sv_ddU
44. Overview | Kernel Memory - Microsoft Open Source, accessed on April 10, 2025, https://microsoft.github.io/kernel-memory/
45. microsoft/kernel-memory: RAG architecture: index and query any data using LLM and natural language, track sources, show citations, asynchronous memory patterns. - GitHub, accessed on April 10, 2025, https://github.com/microsoft/kernel-memory
46. Azure Semantic Kernel Example with Embeddings and Pinecone - Stack Overflow, accessed on April 10, 2025, https://stackoverflow.com/questions/76417790/azure-semantic-kernel-example-with-embeddings-and-pinecone
47. Making AI powered .NET apps more consistent and intelligent with Redis | Semantic Kernel, accessed on April 10, 2025, https://devblogs.microsoft.com/semantic-kernel/making-ai-powered-net-apps-more-consistent-and-intelligent-with-redis/
48. Semantic Kernel Planner 101 - baeke.info, accessed on April 10, 2025, https://blog.baeke.info/2023/06/01/semantic-kernel-planner-101/
49. Introduction to Semantic Kernel Planners for Seamless Orchestration | by Akshay Kokane, accessed on April 10, 2025, https://medium.com/@akshaykokane09/empowering-ai-with-semantic-kernel-planners-for-seamless-orchestration-1c7ad35f2337
50. How to create a GenAI agent using Semantic Kernel | Nearform, accessed on April 10, 2025, https://nearform.com/digital-community/how-to-create-a-genai-agent-using-semantic-kernel/

51. Semantic Kernel Planner 101 - baeke.info, accessed on April 10, 2025, https://baeke.info/2023/06/01/semantic-kernel-planner-101/
52. Semantic Kernel - Sequential Planner - Microsoft Developer Blogs, accessed on April 10, 2025, https://devblogs.microsoft.com/semantic-kernel/semantic-kernel-planners-sequential-planner/
53. How to create a GenAI agent using Semantic Kernel | Nearform, accessed on April 10, 2025, https://www.nearform.com/digital-community/how-to-create-a-genai-agent-using-semantic-kernel/
54. Using Planners in the SK Java Kernel | Semantic Kernel - Microsoft Developer Blogs, accessed on April 10, 2025, https://devblogs.microsoft.com/semantic-kernel/using-planners-in-the-sk-java-kernel/
55. How to create a sequential planner using semantic kernel - YouTube, accessed on April 10, 2025, https://www.youtube.com/watch?v=r-atDSeqLaI
56. Ideas for the planner · microsoft semantic-kernel · Discussion #3429 - GitHub, accessed on April 10, 2025, https://github.com/microsoft/semantic-kernel/discussions/3429
57. Intro to Semantic Kernel – Part Two - Coding, accessed on April 10, 2025, https://blog.brakmic.com/intro-to-semantic-kernel-part-two/
58. Getting Started with Semantic Kernel and C# - Matt on ML.NET - Accessible AI, accessed on April 10, 2025, https://accessibleai.dev/post/introtosemantickernel/
59. Semantic Kernel - Stepwise Planner - Microsoft Developer Blogs, accessed on April 10, 2025, https://devblogs.microsoft.com/semantic-kernel/semantic-kernel-planners-stepwise-planner/
60. Semantic Kernel Stepwise Planner | Restackio, accessed on April 10, 2025, https://www.restack.io/p/semantic-kernel-answer-stepwise-planner-cat-ai
61. Semantic Kernel - Planner - The Developer's Cantina, accessed on April 10, 2025, https://www.developerscantina.com/p/semantic-kernel-planner/
62. 15 - Stepwise Planner in Semantic Kernel #stepwise #planner #semantic - YouTube, accessed on April 10, 2025, https://www.youtube.com/watch?v=a1KhGKsigNo
63. Semantic Kernel Hello World Planners Part 1 - Jason Haley, accessed on April 10, 2025, https://jasonhaley.com/2024/05/19/semantic-kernel-hello-world-planners-part1/
64. What are Planners in Semantic Kernel | Microsoft Learn, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/concepts/planning
65. Semantic Kernel Hello World Planners Part 2 - Jason Haley, accessed on April 10, 2025, https://jasonhaley.com/2024/05/27/semantic-kernel-hello-world-planners-part2/
66. Semantic Kernel - The new planners introduced in 1.0, accessed on April 10, 2025, https://www.developerscantina.com/p/semantic-kernel-new-planners/
67. Semantic Kernel Hello World Plugins Part 1 - Jason Haley, accessed on April 10,

2025,
https://jasonhaley.com/2024/04/11/semantic-kernel-hello-world-plugin-part1/
68. Observability in Semantic Kernel | Microsoft Learn, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/concepts/enterprise-readiness/observability/index
69. Step-by-Step Guide to Building a Powerful AI Monitoring Dashboard with Semantic Kernel and Azure Monitor: Master TokenUsage Metrics and Custom Metrics using SK Filters | by Akshay Kokane | Medium, accessed on April 10, 2025, https://medium.com/@akshaykokane09/step-by-step-guide-to-building-a-powerful-ai-monitoring-dashboard-with-semantic-kernel-and-azure-7d4eed115d31
70. accessed on January 1, 1970, https://learn.microsoft.com/en-us/semantic-kernel/support/SECURITY.md
71. What is Semantic Kernel? Docs, Demo and How to Deploy - Shakudo, accessed on April 10, 2025, https://www.shakudo.io/integrations/semantic-kernel
72. Agentic Workflows with Semantic Kernel on Azure - XenonStack, accessed on April 10, 2025, https://www.xenonstack.com/blog/agentic-workflows-semantic-kernel-azure
73. Semantic Kernel: Fundamentals - Oktay Burak Ertas - Medium, accessed on April 10, 2025, https://oktay-burak-ertas.medium.com/semantic-kernel-fundamentals-5c6a53005a3c
74. Semantic Kernel: Microsoft Answers Your Questions, Updates Docs for AI Integration SDK, accessed on April 10, 2025, https://visualstudiomagazine.com/Articles/2023/06/26/semantic-kernel-qa.aspx
75. Security | Microsoft Learn, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/support/security
76. Making Plans with Semantic Kernel: Setting Up & Authentication - Microsoft Developer Blogs, accessed on April 10, 2025, https://devblogs.microsoft.com/semantic-kernel/making-plans-with-semantic-kernel-setting-up-authentication/
77. NL2SQL "Best Practices" · microsoft semantic-kernel · Discussion #3322 - GitHub, accessed on April 10, 2025, https://github.com/microsoft/semantic-kernel/discussions/3322
78. Need Best practices example setting up semantic kernel with Microsoft Bot Framework #8361 - GitHub, accessed on April 10, 2025, https://github.com/microsoft/semantic-kernel/discussions/8361
79. Semantic Kernel - The basics - The Developer's Cantina, accessed on April 10, 2025, https://www.developerscantina.com/p/semantic-kernel-basics/
80. Semantic Kernel Filters | Microsoft Learn, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/concepts/enterprise-readiness/filters
81. Semantic Kernel Agent Architecture | Microsoft Learn, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/frameworks/agent/agent-architecture
82. The Evolution of LLM Serving: Modern Architectures and Framework Selection -

Adyog, accessed on April 10, 2025, https://blog.adyog.com/2025/02/07/the-evolution-of-llm-serving-modern-architectures-and-framework-selection/
83. MultiCloud Strategy, Deployment and Management - XenonStack, accessed on April 10, 2025, https://www.xenonstack.com/blog/multicloud-strategy
84. AI Agent Deployment: Overcoming Technical Challenges - Rapid Innovation, accessed on April 10, 2025, https://www.rapidinnovation.io/post/for-developers-technical-challenges-and-solutions-in-ai-agent-deployment
85. Components Of AI-Ready Infrastructure In Microsoft Azure - Build5Nines, accessed on April 10, 2025, https://build5nines.com/build-an-ai-ready-infrastructure-in-microsoft-azure/
86. Microsoft.SemanticKernel.Plugins.Document 1.45.0-alpha - NuGet Gallery, accessed on April 10, 2025, https://www.nuget.org/packages/Microsoft.SemanticKernel.Plugins.Document/
87. In-depth Semantic Kernel Demos | Microsoft Learn, accessed on April 10, 2025, https://learn.microsoft.com/en-us/semantic-kernel/get-started/detailed-samples