Capstone Reflection Report

Summary of the AI Agent's Purpose and Functionality

For the Capstone AI Agents Hackathon, We created a simple AI conversational assistant using the Semantic Kernel framework and integrated it with DialoGPT, a pre-trained chatbot model from Hugging Face. The purpose of this agent was to simulate a natural language conversation for educational purposes. It could respond to general questions and provide summaries or brief explanations of input text.

The agent was designed to demonstrate how large language models can be used in combination with Semantic Kernel to build modular, extensible AI systems. Using Semantic Kernel allowed us to structure the assistant with reusable skills and prompts, enabling better separation of concerns and clarity in how the AI responded.

Key Challenges and How I Overcame Them

One of the biggest challenges We faced was working with the Semantic Kernel in a Google Colab environment. Since the documentation and support for Semantic Kernel in Colab were limited, several issues occurred, especially around installing the correct versions of packages and finding the right functions (e.g., errors like create_function_from_prompt not being found).

To overcome these hurdles:

- We carefully followed the Semantic Kernel GitHub documentation and cross-checked examples from Microsoft Learn.
- We downgraded or upgraded packages when necessary to avoid compatibility issues (like API deprecation in openai).
- We isolated functionality, testing smaller components before integrating them into the main agent.
- We also switched temporarily to using just Hugging Face's transformers to validate our chatbot approach before layering in Semantic Kernel for structure.

Another challenge was managing API rate limits and quota issues with OpenAI. When we hit a quota error, We pivoted to using free Hugging Face models like microsoft/DialoGPT-small, which allowed us to continue development without interruptions.

What We Learned from Microsoft Learn and Hackathon Sessions

The Microsoft Learn module**s** on Semantic Kernel were incredibly valuable for understanding:

- How to organize AI functions as plugins or skills.
- The importance of separating the orchestration layer from the model itself.
- How prompt engineering fits into a modular AI agent pipeline.

We learned how Semantic Kernel can bridge traditional code with LLM capabilities, offering a powerful way to scale and maintain conversational agents with clarity.

The Hackathon sessions helped us connect theory to practice. Seeing how other teams approached similar problems gave me new ideas for how to improve our agent. For example, one team demoed using plugins in a smart way that inspired me to refactor my summarization logic into its own function.

Hackathon Experience

The hackathon experience was both challenging and rewarding. Time management was critical, especially as we balanced debugging issues with creating a working demo. We allocated our time across planning (30%), coding (50%), and testing/refinement (20%).

Though we worked independently for the most part, we learned a lot from interacting with our peers and instructors. If we had more time, we would have liked to add a memory feature to the agent so it could recall past conversations and build context.

Overall, this hackathon gave us confidence in building AI tools using real-world frameworks and helped me feel more prepared for future projects, especially those involving LLMs and modular architecture.

Conclusion

This Capstone project pushed us to go beyond basic chatbot implementations and dive into frameworks like Semantic Kernel that encourage scalable design. Despite technical setbacks and quota limits,We were able to build a working AI agent that uses LLMs to simulate conversation, learning valuable lessons in AI design, prompt structuring, and tool integration.

This experience not only fulfilled the project goals but also reinforced our interest in AI and conversational systems. We now feel more capable and inspired to build more advanced AI agents in the future.

The Three Live Sessions

Build your code-first app with Azure AI Agent Service

In this session of the AI Agents Hackathon, I focused on building AI agents in Azure using the Azure AI Agent Service. The speaker introduced the service, which simplifies the creation of agent-based applications, and shared an example use case for an e-commerce company, "Kontoso," that analyzes sales data for its staff. I learned about the concept of an AI agent, including its semi-autonomous nature, its ability to reason over business processes, and the importance of giving it access to various data sources. In the demo, I saw how to use a conversational UI framework (Chainlit), where the AI agent answers user queries, generates dynamic SQL for database searches, creates data visualizations, and exports data to Excel. The session also covered important security precautions for running agents that interact with databases and external systems.

Prototyping AI Agents with GitHub Models

In this session, I learned how large language models (LLMs) can recommend calling external functions, also known as tools, during conversations. The LLM suggests which tool to use and what arguments to send, but it's still up to me to decide whether or not to actually call the function. This idea is the foundation for building AI agents, where tools let the agent take actions or look up information. After understanding that, we moved into Python AI agent frameworks. We were experimenting with four frameworks: OpenAI Agents (a newer, more flexible SDK), Pydantic-AI, Autogen, and Semantic Kernel. All of these can work with GitHub models, Azure OpenAI, or even local servers like Ollama by just adjusting the API base URL and key. We focused first on OpenAI Agents, which is model-agnostic and designed specifically for agent workflows, offering more flexibility than the regular OpenAI package. There’s also a public GitHub repo with all the examples so I can follow along and practice.

Building smarter Python AI agents with code interpreters

In this session, I listened to a live coding-focused livestream where the presenter explored how to make AI agents smarter by adding code interpreters. They started with a simple retrieval-augmented generation (RAG) app using Python, LangChain, Chainlit for the chat UI, and Azure OpenAI, showing how it retrieved information from an Azure AI Search vector store filled with documents about Azure Container Apps. After demonstrating that the app could answer basic questions but sometimes hallucinated math results, they added a Python REPL (Read-Eval-Print Loop) tool to the agent, enabling it to correctly compute mathematical operations by writing and executing Python code. The presenter emphasized the risks of giving an agent code execution capabilities, explaining how it could be exploited to access sensitive

files or run harmful commands if deployed publicly. They also briefly mentioned future demos, like building a custom code interpreter server (MCP) to integrate with Cloud Desktop. The session was informal and heavily focused on live coding, with minimal slides.