Empowering .NET Applications with Semantic Kernel and Azure OpenAI

Orestis Meikopoulos

Head of Engineering @ Code Create

https://linkedin.com/in/ormikopo

Agenda

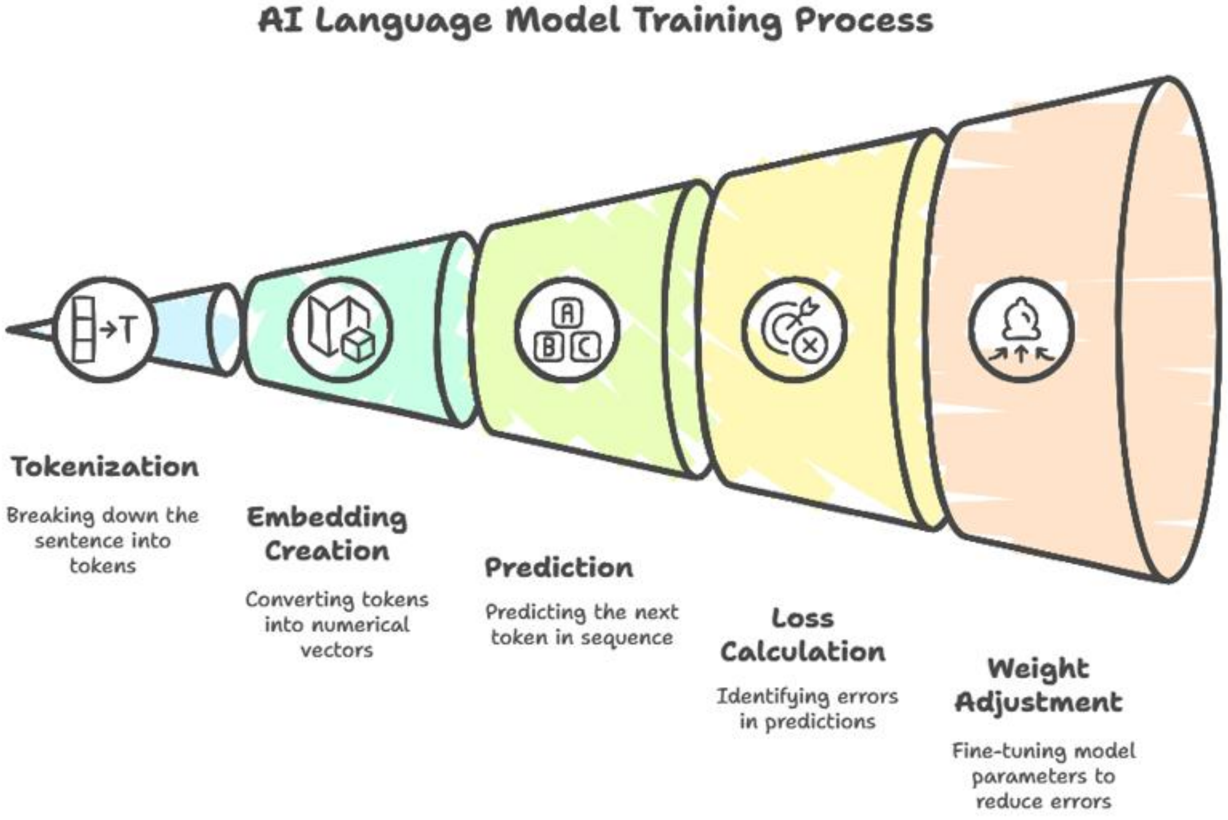
- Key concepts:
  - Generative AI & LLMs
  - Tokens
  - Embeddings
  - Vector databases
  - Retrieval Augmented Generation
  - OpenAI function calling
- Semantic Kernel
- Azure AI Agent Service
- Demo

How Generative AI & LLMs Work

- **Generative AI in a nutshell:** Creates new text, code, images or audio from a user prompt instead of just classifying or searching.
- **Everything starts with a model**: Trained on huge datasets, spots patterns and produces statistically similar but fresh output.
- **LLMs - The language engine**: A Generative AI app uses an LLM for natural-language I/O; other specialist models handle images or audio.

How Generative AI & LLMs Work

- **Under the hood - The token loop**:
  - Prompt is broken into tokens (word pieces).
  - Each token gets an embedding (aka, a high-dimensional meaning vector).
  - The model predicts the next token, one step at a time, updating like super-powered autocomplete.
- **Learning = minimizing "oops"**: During training the model sees the real next token, measures the error (loss) and nudges its weights to do better (millions of times over).
- **What you can build**: Natural-language generation, image, audio, and code creation (e.g., ChatGPT writes a C# tic-tac-toe game).



AI Language Model Training Process

Tokenization

Breaking down the sentence into tokens

Embedding Creation

Converting tokens into numerical vectors

Prediction

Predicting the next token in sequence

Loss Calculation

Identifying errors in predictions

Weight Adjustment

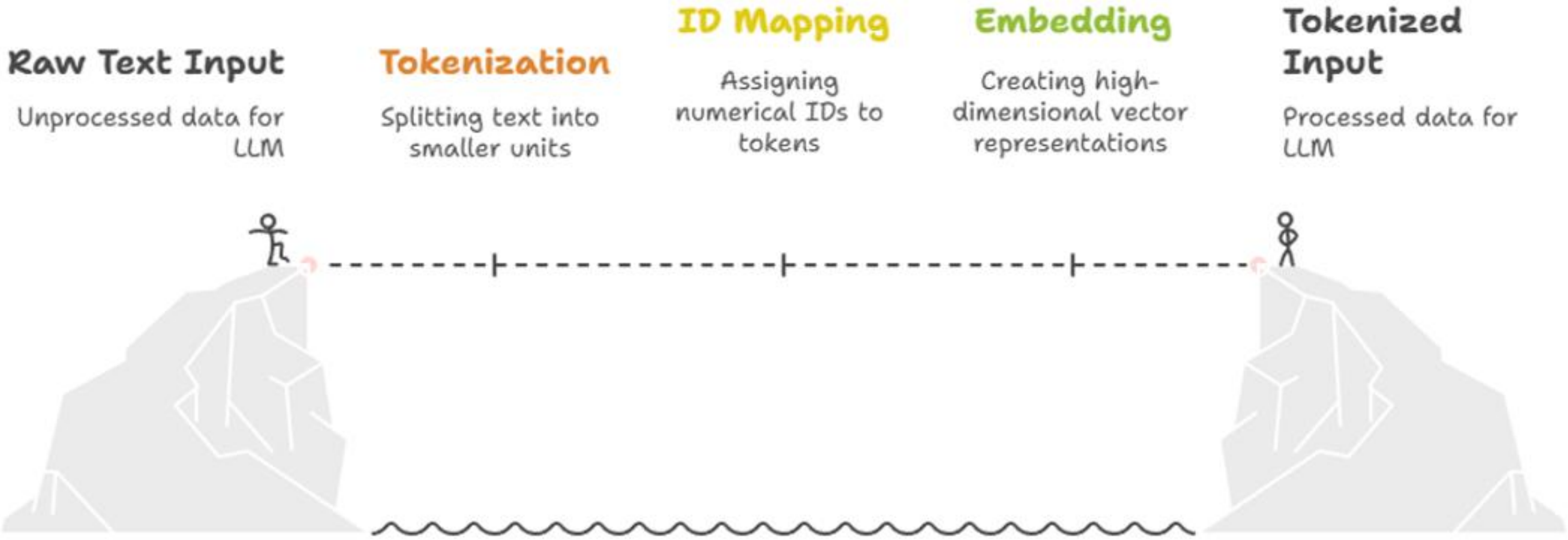
Fine-tuning model parameters to reduce errors

Tokens: The building blocks of LLMs

- **What’s a token**: The smallest chunk of text an LLM understands. It could be a whole word, part-word, punctuation, or even a single character.
- **Tokenization = step #1**:
  - Every prompt and training document is split into tokens, forming the model’s vocabulary.
  - Example: I heard a dog bark loudly at a cat → 9 tokens.
- **Three common tokenization methods**: Word, character, sub-word. Each trades flexibility vs. efficiency.

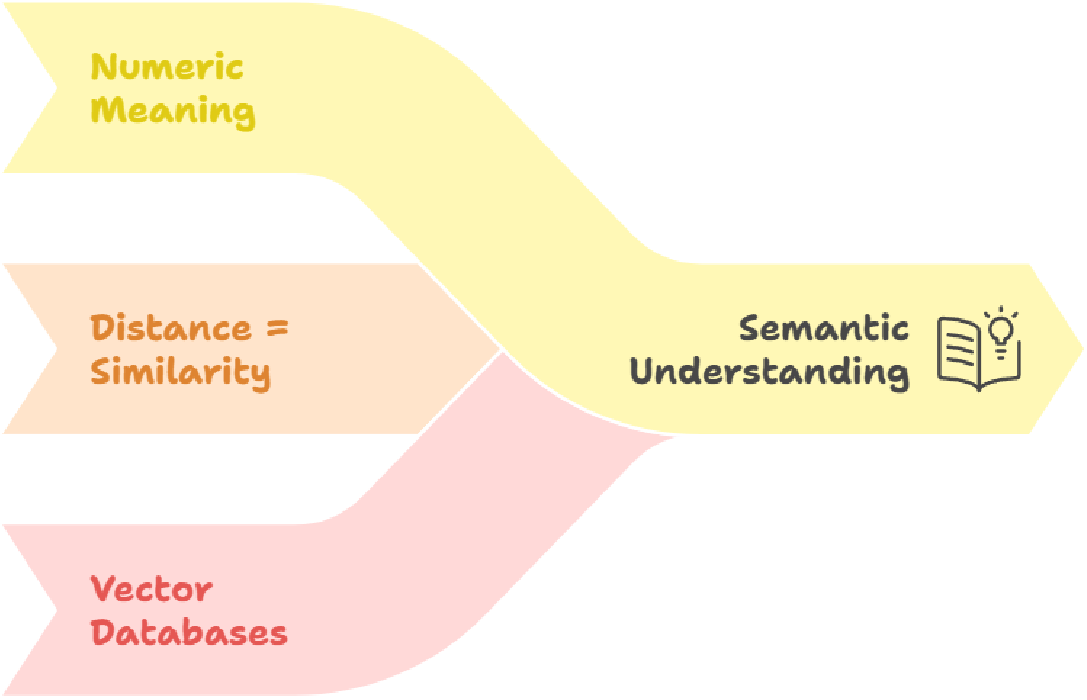
Tokens: The building blocks of LLMs

- **Token IDs & embeddings**: Tokens are mapped to numeric IDs, then to high-dimensional embeddings that capture meaning and context.
- **Context window & limits**: Every model has a max number of input + output tokens (the context window). Go over and your request is clipped or rejected.
- **Why developers should care**: Latency, cost, and rate limits are all measured in tokens, not characters. Concise prompts save money and time.


Raw Text Input
Unprocessed data for LLM
Tokenization
Splitting text into smaller units
ID Mapping
Assigning numerical IDs to tokens
Embedding
Creating high-dimensional vector representations
Tokenized Input
Processed data for LLM

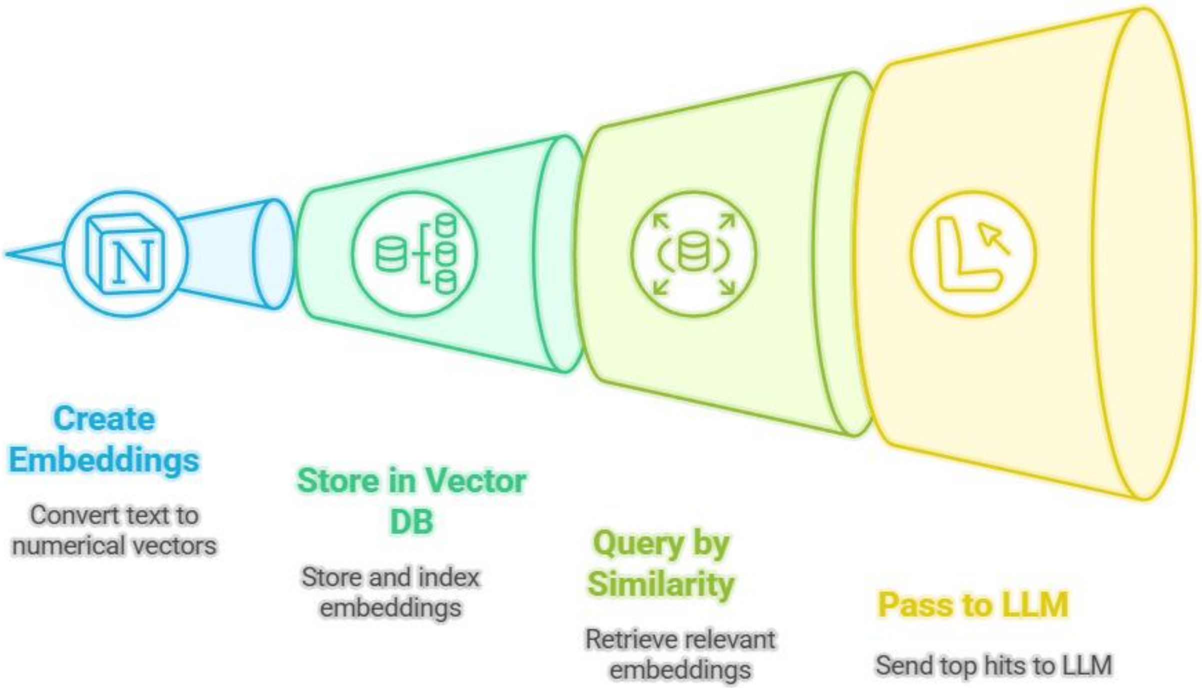
Embeddings: Giving AI a sense of meaning

- **Numeric meaning**: An embedding is a dense vector that captures the semantic essence of text, code, audio, or images.
- **Distance = similarity**: Vectors that point in a similar direction mean their originals are concept-wise close. Cosine similarity is the usual yard-stick.
- **Where they live**: Store millions of vectors in a *vector database*, like Azure AI Search, Cosmos DB, Postgres, etc.
- **Why developers should care**:
  - Retrieval-augmented generation (RAG) with your data.
  - Summaries & prompt compression to fit token limits
  - Classification, translation, recommendation, multimodal mash-ups, etc.



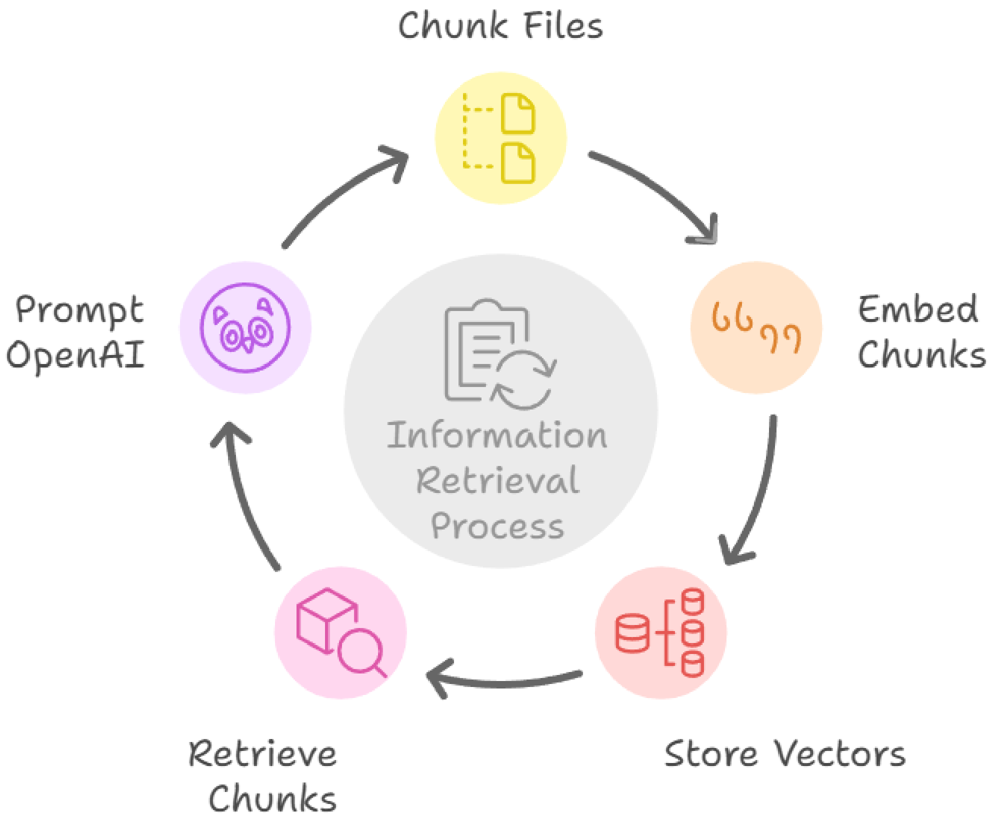
Vector Databases: Long-term memory for your .NET AI apps

- **What they are:** Databases built to *store & index* high-dimensional **embeddings** so you can search by *meaning* instead of keywords.
- **Why we use them:** Similar-item discovery, product or content recommendations, anomaly & fraud detection, and **Retrieval-Augmented Generation (RAG)** for LLM chat with your own data.
- **Typical vector search workflow**:
  - Create embeddings with Azure OpenAI.
  - Store / index in a vector DB (Azure AI Search, Cosmos DB, etc.).
  - Convert user query to an embedding, then run a **nearest-neighbor search** (cosine/Euclidean) inside vector database and query by similarity.
  - Pass top hits to LLM.


Create Embeddings
Convert text to numerical vectors
Store in Vector DB
Store and index embeddings
Query by Similarity
Retrieve relevant embeddings
Pass to LLM
Send top hits to LLM

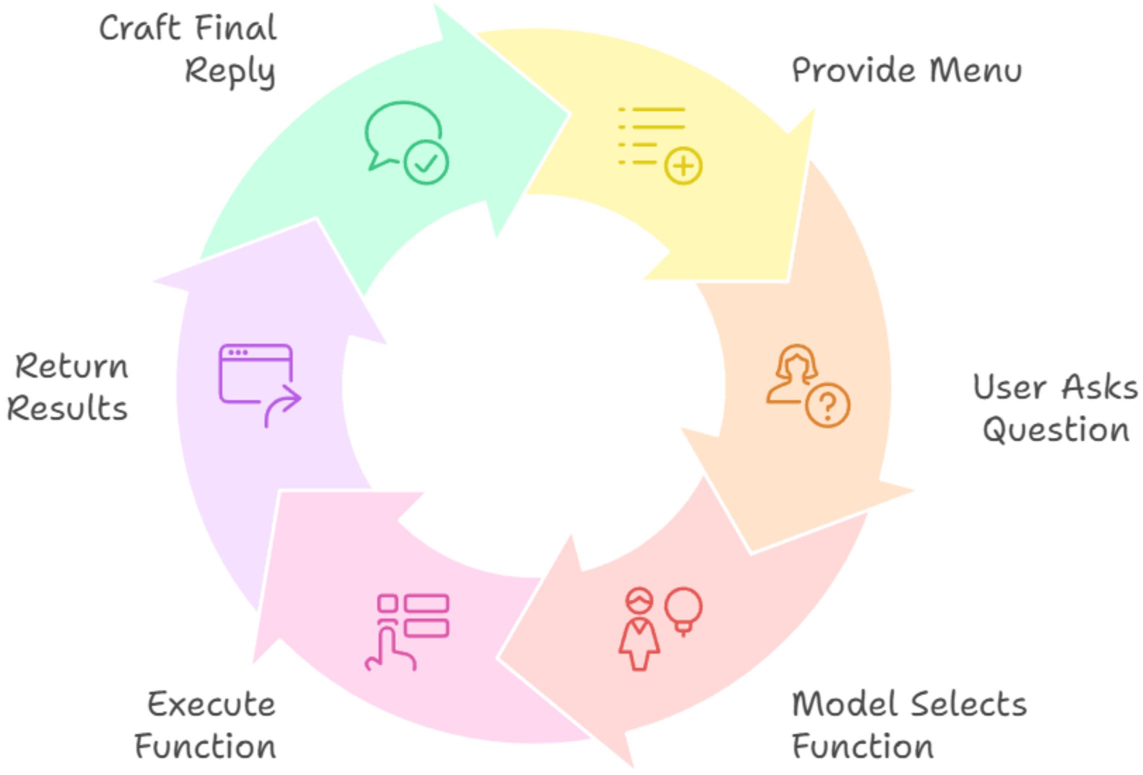
Retrieval-Augmented Generation (RAG)

- Brings your own data into the conversation without re-training the LLM.
- Workflow: Chunk → Embed → Store → Retrieve → Prompt.
- Uses vector search to find the most relevant pieces of content.
- Cheaper & faster than fine-tuning. Keeps answers fresh and source-linked.
- Easily wired into .NET with Semantic Kernel + Azure AI Search / Cosmos DB.



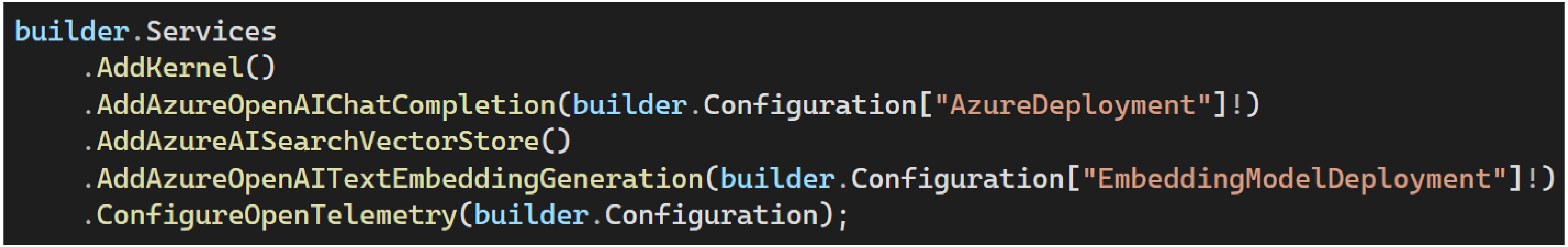
OpenAI Function Calling

- **Describe each tool as JSON**: The model returns which function to run + arguments instead of pure text.
- **Turns the LLM into a router**: For real-time APIs, databases, or code. No extra fine-tuning needed.
- **Workflow**: Prompt → JSON call(s) → execute → feed results → final answer.
- Built into Semantic Kernel with **[KernelFunction]** attributes on C# methods



Semantic Kernel’s “Kernel” (The mission-control center)

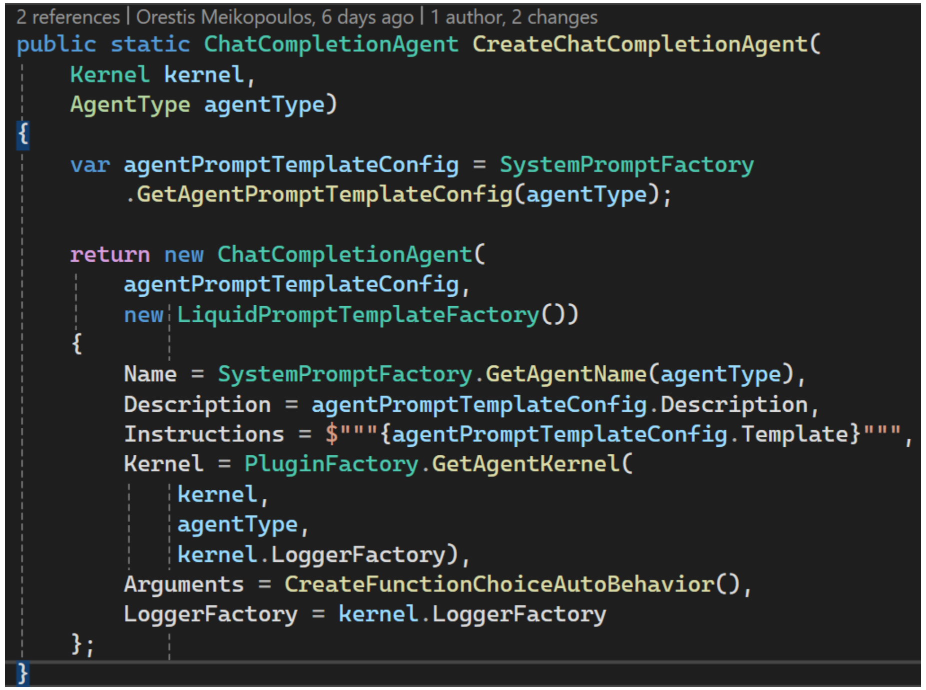
- **Central DI container**: Holds every AI service & plugin your app needs.
- **Orchestrates the whole cycle**: Pick service → build prompt → send → parse → return.
- **Key Semantic Kernel components**
  - AI service connectors
  - Vector-store (aka. memory) connectors
  - Functions & Plugins
  - Prompt templates
  - Filters



```
builder.Services
     .AddKernel()
     .AddAzureOpenAIChatCompletion(builder.Configuration["AzureDeployment"]!)
     .AddAzureAISearchVectorStore()
     .AddAzureOpenAITextEmbeddingGeneration(builder.Configuration["EmbeddingModelDeployment"]!)
     .ConfigureOpenTelemetry(builder.Configuration);
```

Semantic Kernel Agent Framework

- Adds autonomous, goal-driven agents on top of familiar Semantic Kernel patterns.
- Base **Agent** class: Concrete types like **ChatCompletionAgent**, **AzureAIAgent**, **OpenAIAssistantAgent** (all kernel-powered).
- Agents can co-operate in an **AgentChat** / **AgentGroupChat** for multi-step or multi-agent workflows.
- Each agent links to a Kernel + Plugins / Function Calling: Real tools & memory inside the loop.



2 references | Orestis Meikopoulos, 6 days ago | 1 author, 2 changes

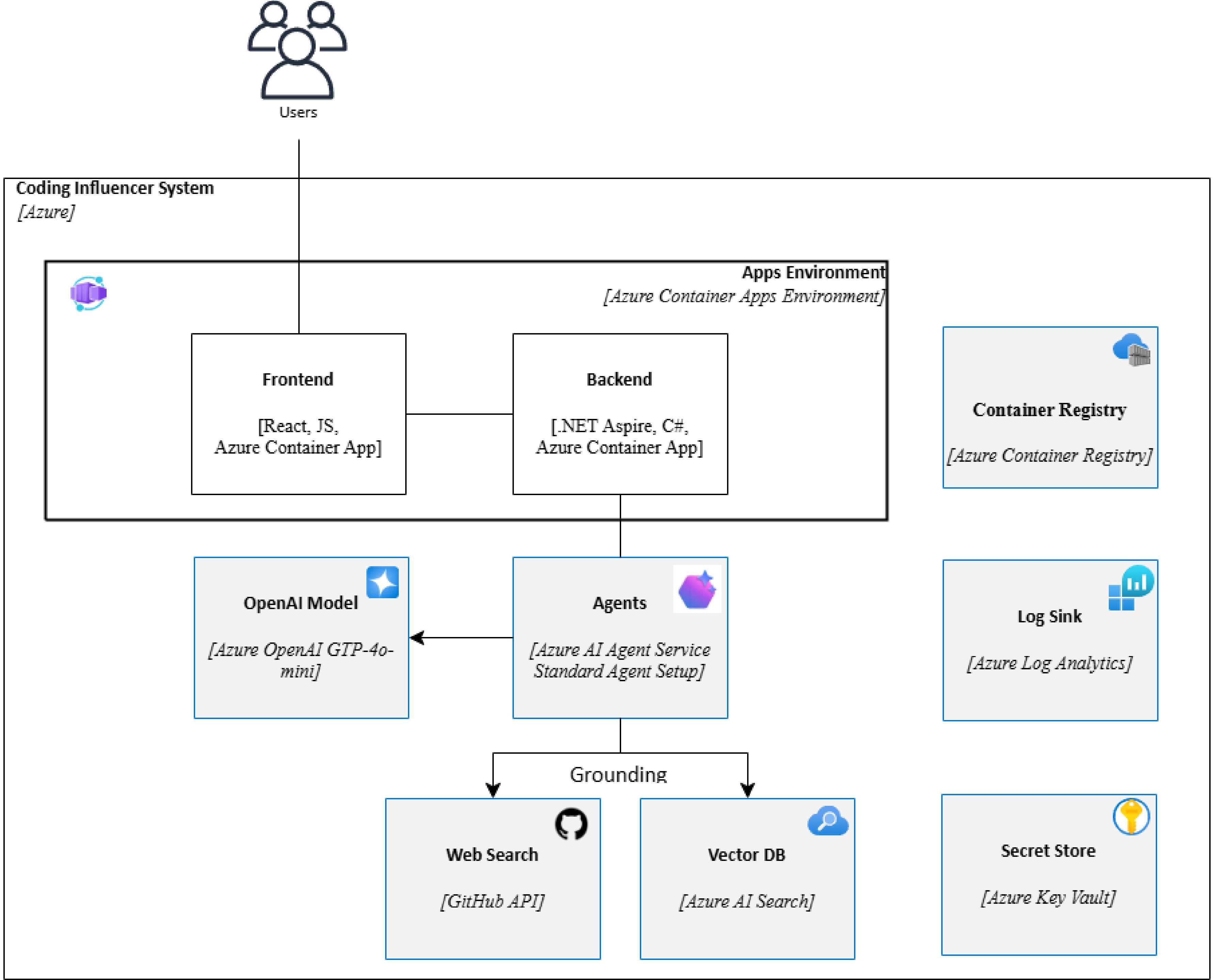
```
public static ChatCompletionAgent CreateChatCompletionAgent(
    Kernel kernel,
    AgentType agentType)
{
    var agentPromptTemplateConfig = SystemPromptFactory
        .GetAgentPromptTemplateConfig(agentType);

    return new ChatCompletionAgent(
        agentPromptTemplateConfig,
        new LiquidPromptTemplateFactory())
    {
        Name = SystemPromptFactory.GetAgentName(agentType),
        Description = agentPromptTemplateConfig.Description,
        Instructions = $"""{agentPromptTemplateConfig.Template}""",
        Kernel = PluginFactory.GetAgentKernel(
            kernel,
            agentType,
            kernel.LoggerFactory),
        Arguments = CreateFunctionChoiceAutoBehavior(),
        LoggerFactory = kernel.LoggerFactory
    };
}
```

Azure AI Agent Service: Managed, server-side Agents

- **Fully managed**: Build, deploy, and scale AI agents without running any infrastructure yourself.
- **Server-side tool calling & state**: Azure handles the entire JSON-call → invoke → response loop and persists conversation threads.
- **Rich built-in tools**: File Search, Code Interpreter, Bing, Azure AI Search, Azure Functions & bring your own.
- **Model choice freedom**: Mix Azure OpenAI, Llama 3, Mistral, under the same API.
- **Enterprise extras**: Keyless auth, BYO or managed storage, Responsible-AI filters, elastic scaling.

Demo



Users

Coding Influencer System
[Azure]

Apps Environment
[Azure Container Apps Environment]

Frontend

[React, JS,
Azure Container App]

Backend

[.NET Aspire, C#,
Azure Container App]

Container Registry

[Azure Container Registry]

OpenAI Model

[Azure OpenAI GTP-4o-mini]

Agents

[Azure AI Agent Service Standard Agent Setup]

Log Sink

[Azure Log Analytics]

Grounding

Web Search

[GitHub API]

Vector DB

[Azure AI Search]

Secret Store

[Azure Key Vault]

Thank you

- For the opportunity
- For participating
- For listening