Herding AI Cats

TAMING SEMANTIC KERNEL MULTI-AGENT SCENARIOS

Marc Plogas

Software Engineering
Cloud Architecture
IoT
ML / AI

The Hook: Herding Cats

The Problem: Orchestrating Collaboration

Challenge 1: Task Decomposition and Specialization

Challenge 2: State Management and Communication

Challenge 3: Control Flow

Challenge 4: Tool Integration

The Solution: Semantic Kernel Agents

Semantic Kernel

Agent Framework

- ChatCompletionAgent
- AgentGroupChat
- Prompt-Driven Orchestration

Session Objectives

Understand: Core Semantic Kernel concepts (Agents, Plugins)

See: A practical multi-agent scenario in action

Learn: How configuration can drive agent behaviour and orchestration

Explore: Key code components for building and running this scenario

What to expect?

Intro: Semantic Kernel, Agents and Plugins

Demo

Code Deep-Dive

Q&A

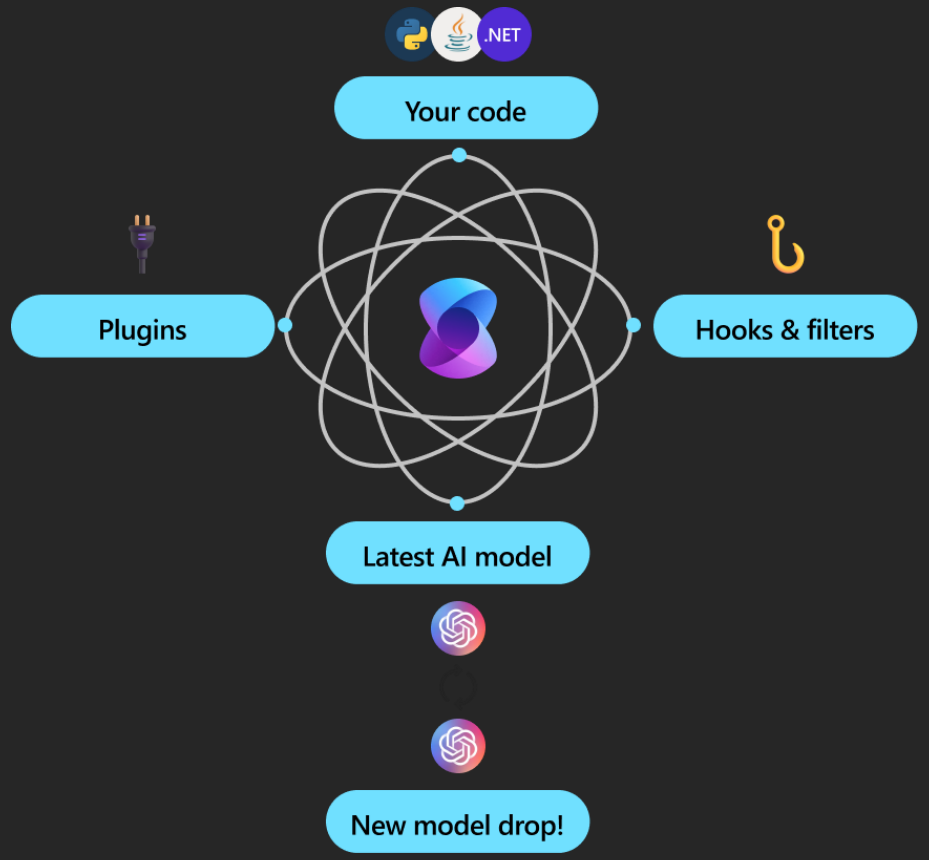
Code: https://github.com/mplogas/sk-multiagent

INTRO: SEMANTIC KERNEL, AGENTS AND PLUGINS

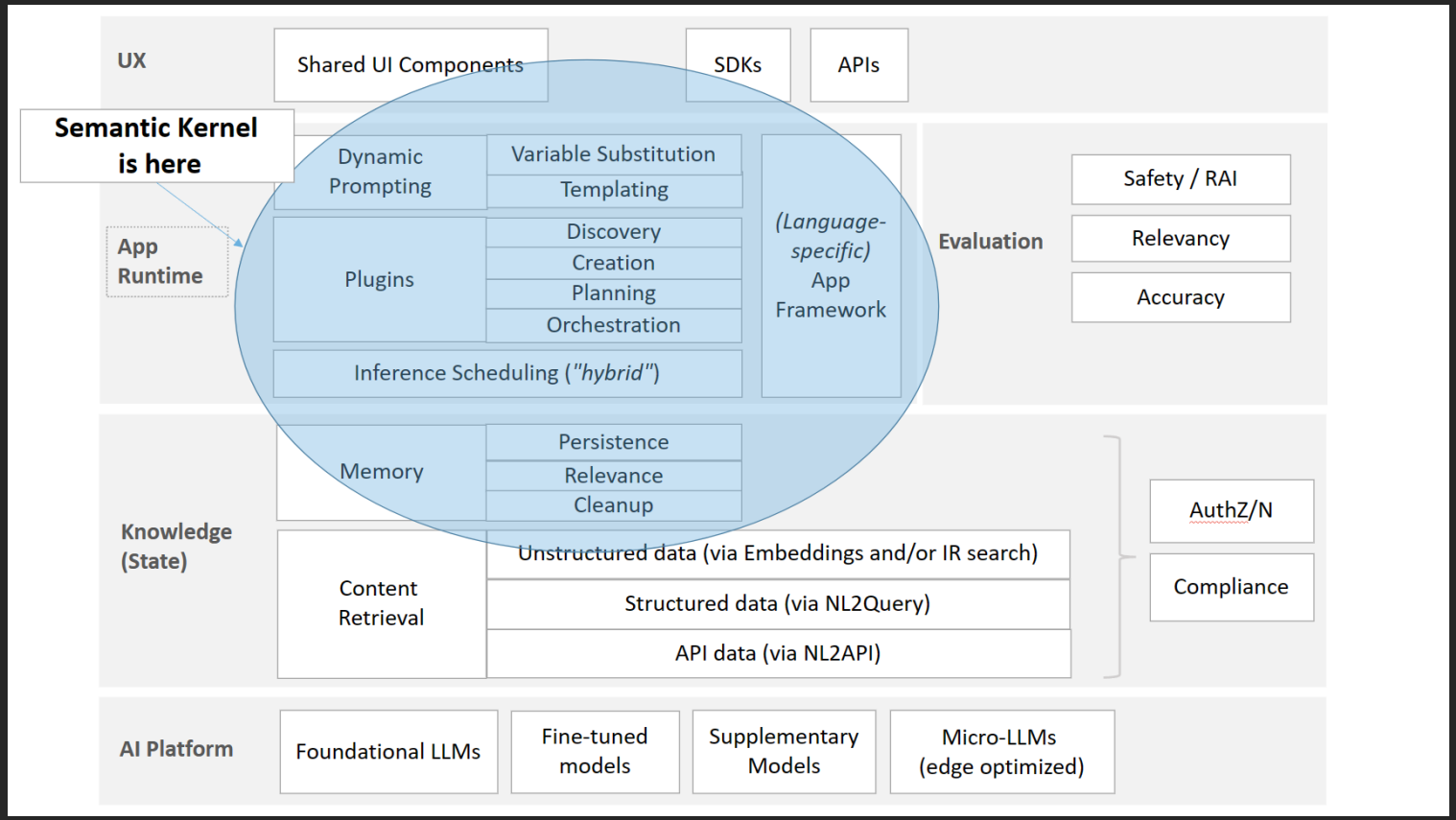
What is Semantic Kernel

Open-Source SDK

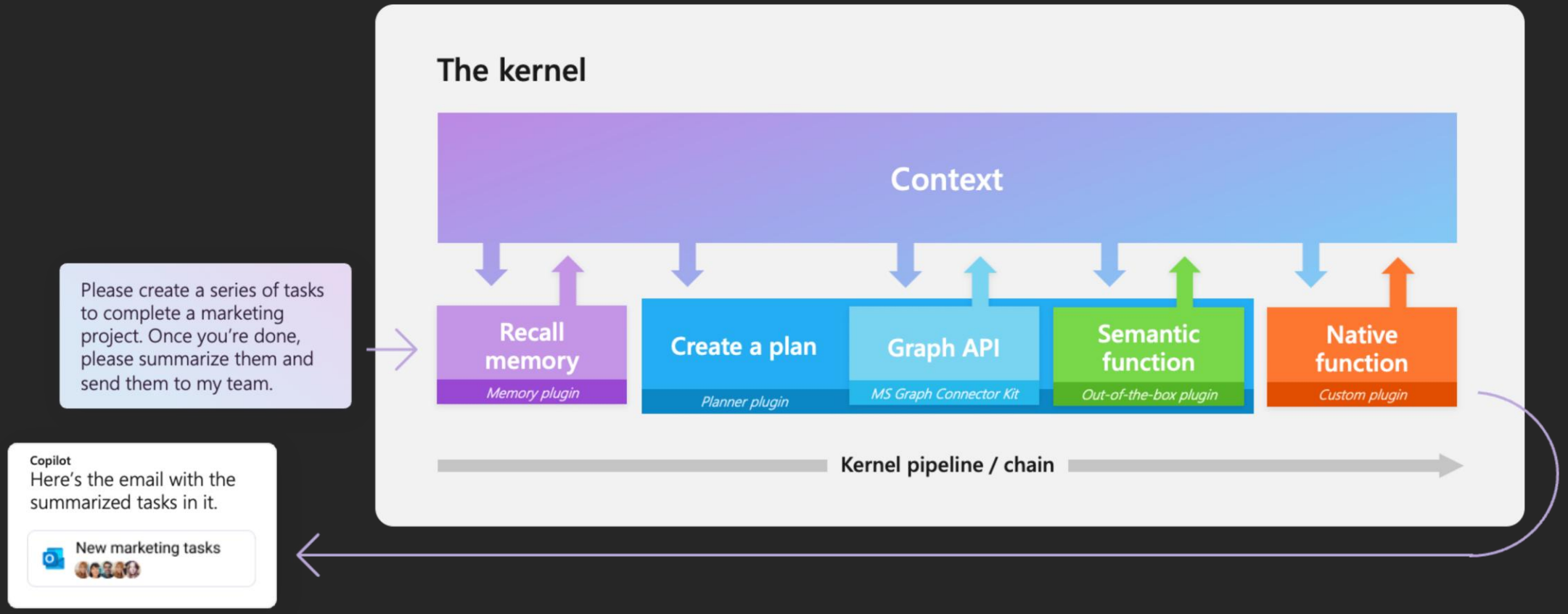
Orchestration Engine



What is Semantic Kernel



Core Components: Kernel



Core Components: Prompt Templates

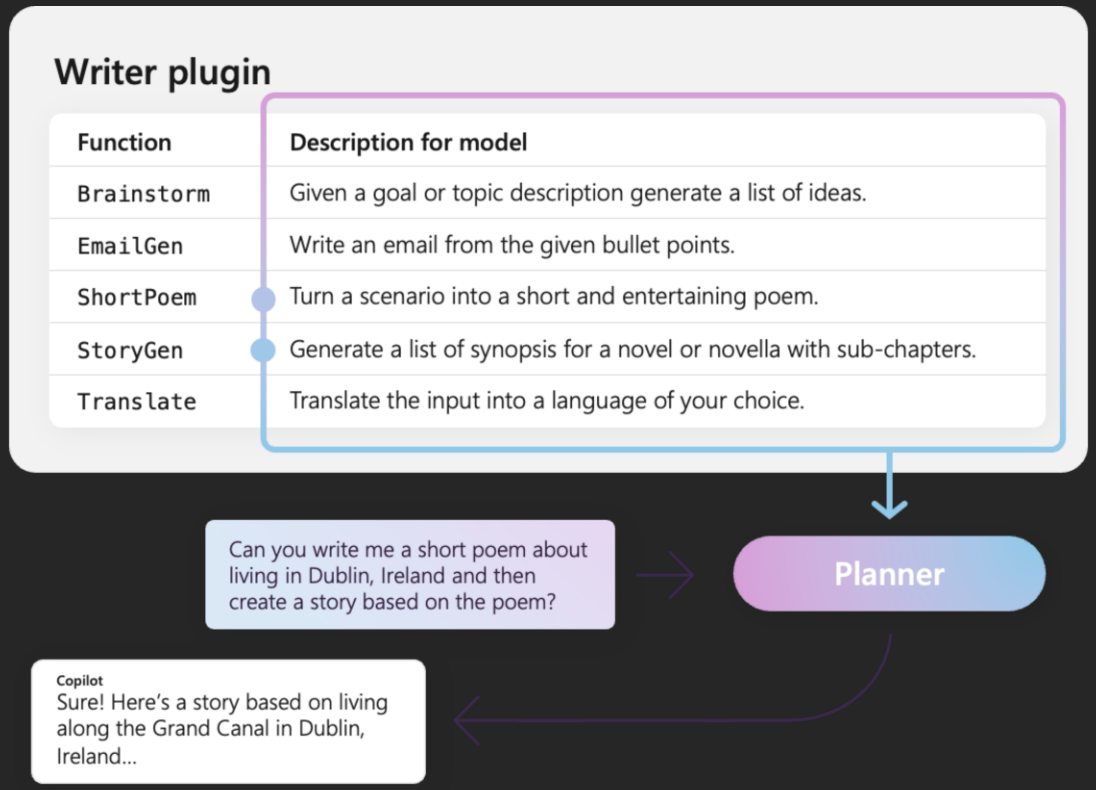
Variables

```
{{set "userName" "Alice"}}
Hello, {{userName}}! How can I assist you today?
```

External Function calling and passing parameters

```
{{set "number" 10}}
The result of doubling {{number}} is
{{MathPlugin.Multiply number=number factor=2}}.
```

Core Components: Plugins and Functions



Core Components: Memory

Feature	Semantic Memory	Kernel Memory
Data formats	Text only	Web pages, PDF, Images, Word, PowerPoint, Excel, Markdown, Text, JSON, more being added
Search	Cosine similarity	Cosine similarity, Hybrid search with filters, AND/OR conditions
Language support	C#, Python, Java	Any language, command line tools, browser extensions, low-code/no-code apps, chatbots, assistants, etc.
Storage engines	Azure AI Search, Chroma, DuckDB, Kusto, Milvus, MongoDB, Pinecone, Postgres, Qdrant, Redis, SQLite, Weaviate	Azure AI Search, Elasticsearch, Postgres, Qdrant, Redis, SQL Server, In memory KNN, On disk KNN. In progress: Chroma

Introducing Agents

- Software components that leverage AI models to do work
  - e.g. a chat agent (aka chatbot) that searches your own data and answers questions around it
  - a copilot that is helping you with tasks such as composing emails based on bullet points or notes
- Fully autonomous or work alongside you
- You can specialize them to perform a specific task, making them more effective

The ChatCompletionAgent

```
// Initialize a Kernel with a chat-completion service
IKernelBuilder builder = Kernel.CreateBuilder();

builder.AddAzureOpenAIChatCompletion(/*<...configuration parameters>*/);

Kernel kernel = builder.Build();

// Create the agent
ChatCompletionAgent agent =
    new()
    {
        Name = "SummarizationAgent",
        Instructions = "Summarize user input",
        Kernel = kernel
    };
```

Multi-Agent Collaboration: AgentGroupChat

```
AgentGroupChat chat = new(travelManager, travelAgent, flightAgent)
{
    ExecutionSettings = new()
    {
        TerminationStrategy = new KernelFunctionTerminationStrategy(terminateFunction, kernel)
        {
            Agents = [travelManager],
            ResultParser = (result) => result.GetValue<string>()?.Contains("yes", StringComparison.OrdinalIgnore
            HistoryVariableName = "history",
            MaximumIterations = 10
        },
        SelectionStrategy = new KernelFunctionSelectionStrategy(selectionFunction, kernel)
        {
            AgentsVariableName = "agents",
            HistoryVariableName = "history"
        }
    }
};
```

DEMO SCENARIO: WHO NEEDS HUMANS ANYWAY?

The Scenario: Automating Feature Development

Goal: Simulate a simplified software development lifecycle using collaborating AI agents

Input: A user's feature request (natural language prompt).

Process: Agents work sequentially (and sometimes iteratively) to:

- Define Requirements
- Implement Code
- Review Code
- Generate Documentation

Output: Requirements, Code, and Documentation files (.md) saved to the filesystem

Meet the (AI) Team pt.1

Requirements Engineer:

- **Role**: Analyzes user input, creates detailed functional/non-functional requirements & acceptance criteria
- **Output**: requirements.md

Senior Developer:

- **Role**: Reads requirements, writes C# code & basic unit tests, adheres to best practices
- **Output**: code.md

Meet the (AI) Team pt.2

Code Reviewer:

- **Role**: Evaluates the SeniorDeveloper's code against requirements, quality standards (completeness, best practices, performance, security). Provides ratings (0-5) and feedback
- **Output**: Approves code (if scores ≥ 4) or requests revisions

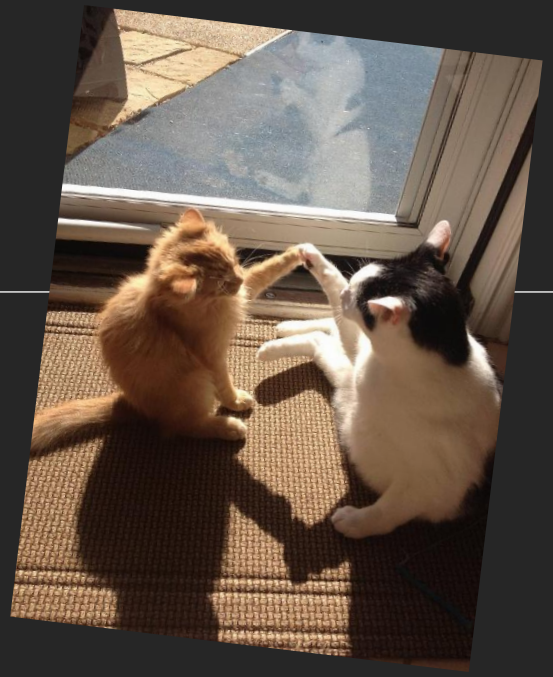
Documentation Specialist:

- **Role**: Takes approved code, generates API docs, user manual snippets, and changelog entries
- **Output**: documentation.md and signals SUCCESS for termination if documentation is completed

The Shared Tool: FileSystemPlugin

```
public class FileSystemPlugin
{
    private readonly string _allowedBaseDirectoryFullPath;

    // Constructor accepting the configured base path
    public FileSystemPlugin(string configuredBasePath)
    {…
    }

    [KernelFunction, Description("Reads the entire content of a specified file.")]
    public async Task<string> ReadFileAsync(
        [Description("The full path to the file to read.")] string filePath)
    {…
    }

    [KernelFunction, Description("Writes the given content to a specified file. Overwrites the file if it already exists.")]
    public async Task<string> WriteFileAsync(
        [Description("The full path to the file to write.")] string filePath,
        [Description("The content to write to the file.")] string content)
    {…
    }

    private bool IsPathAllowed(string requestedPath)
    {…
    }
}
```

The Workflow: Cats Can Collaborate



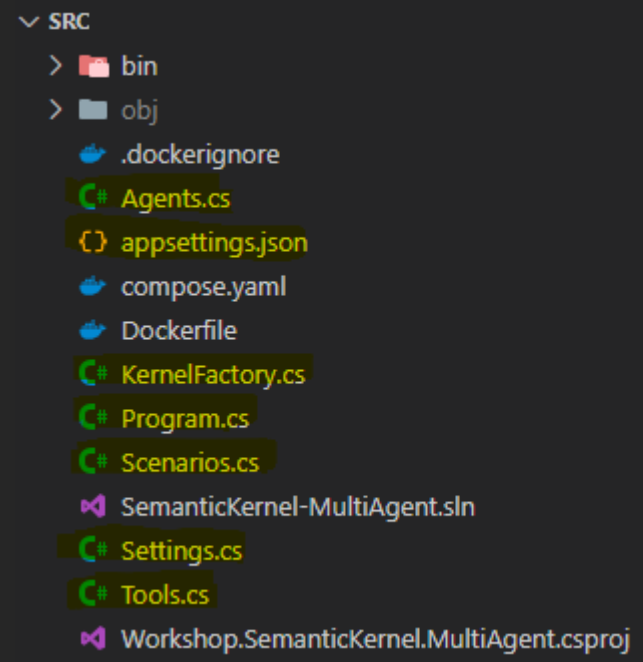
DEMO, FOR REAL! (PRAY TO THE DEMO GODS!)

THE BETTER PART: CODE

Overview: Structure and Classes

Core Classes:

- Settings
- KernelFactory
- ToolFactory / FilesystemPlugin
- Agents
- Scenarios
- Program



Agent Definition: appsettings.json

(Code snippet showing one agent, e.g RequirementsEngineer)

```
{
    "name": "RequirementsEngineer",
    "description": "...",
    "backend": "azureopenai", // Or openai, ollama, gemini
    "model": "gpt-4.5-preview", // Specific model
    "tools": ["FileSystem"], // Tools available
    "instructions": "You are the Requirements Engineer... \nYou have access to a tool named 'FileSystem'... use 'Fi
}
```

Key Fields: **name**, **description**, **backend** / **model**, assigned **tools**, and detailed **instructions** (the agent's persona/prompt).

Scenario Definition: appsettings.json

(Code snippet showing the scenario definition)

```
"scenarios": [
  {
    "enabled": true,
    "name": "development",
    "description": "A user requests a new feature, and the Requirements Engineer converts it into
    "agents": ["RequirementsEngineer", "SeniorDeveloper", "CodeReviewer", "DocumentationSpecialis
    "selectionprompt": "$$$\"\"\"  \nYour job is to determine which participant takes the next tu
    "terminationprompt" : "$$$\"\"\"  \nExamine the HISTORY and determine whether the documentati
    "terminationagents": ["DocumentationSpecialist"],
    "terminationsuccess": "SUCCESS",
    "backend": "azureopenai",
     "model": "o1-mini"
  }
]
```

Key Fields: **enabled**, **name**, **agents,** orchestration prompts, and conditions

Loading Settings

(Code snippet from Settings.cs)

```
public TSettings GetSettings<TSettings>(string name) where TSettings : new()
{
    var section = this._configRoot.GetSection(name);
    return section.Exists() ? section.Get<TSettings>() ?? new TSettings() : new TSettings();
}

public TransformerBackendSettings GetTransformerBackendSettings(TransformerBackend backend)
{
    switch (backend)
    {
        case TransformerBackend.OpenAI:
            return this.GetSettings<OpenAISettings>("OpenAI");
        //other backends
        case TransformerBackend.Ollama:
        default:
            return this.GetSettings<TransformerBackendSettings>("Ollama");
    }
}
```

Provides typed configuration from appsettings/environment variables and user secrets

Creating Kernels

(Code snippet from KernelFactory.cs)

```
public static Kernel CreateKernel(ILoggerFactory loggerFactory, Settings settings, string model, TransformerBackend backend)
{
    var backendSettings = settings.GetTransformerBackendSettings(backend);
    var kernelBuilder = Kernel.CreateBuilder();
    kernelBuilder.Services.AddSingleton(loggerFactory);
    switch (backend)
    {
        case TransformerBackend.OpenAI:
            var oaiSettings = backendSettings as Settings.OpenAISettings;
            return kernelBuilder.AddOpenAIChatCompletion(model, oaiSettings.ApiKey, oaiSettings.Organization).Build();
        case TransformerBackend.AzureOpenAI:
            return kernelBuilder.AddAzureOpenAIChatCompletion(model, backendSettings.Endpoint, backendSettings.ApiKey).Build();
        case TransformerBackend.Ollama:
            return kernelBuilder.AddOllamaChatCompletion(model, new Uri(backendSettings.Endpoint)).Build(); //if you have an api ke
        case TransformerBackend.Gemini:
            return kernelBuilder.AddGoogleAIGeminiChatCompletion(model, backendSettings.ApiKey).Build();
        default:
            throw new ArgumentOutOfRangeException(nameof(backend), backend, null);
    }
}
```

Builds Kernel instances configured for specific AI backends

Kernel Execution Settings

(Code snippet from KernelFactory.cs)

```
public static PromptExecutionSettings GetExecutionSettings(TransformerBackend backend)
{
    switch (backend)
    {
        case TransformerBackend.OpenAI:
        case TransformerBackend.AzureOpenAI:
            return new OpenAIPromptExecutionSettings
            {
                FunctionChoiceBehavior = FunctionChoiceBehavior.Auto()
            };
        case TransformerBackend.Gemini:
            return new GeminiPromptExecutionSettings
            {
                ToolCallBehavior = GeminiToolCallBehavior.AutoInvokeKernelFunctions
            };
        case TransformerBackend.Ollama:
        default:
            return new PromptExecutionSettings()
            {
                FunctionChoiceBehavior = FunctionChoiceBehavior.Auto()
            };
    }
}
```

ExecutionSettings are model-specific and enable the model to automatically call tools when needed.

Defining Tools

(Code snippet from Tools.cs)

```
public class FileSystemPlugin
{
    private readonly string _allowedBaseDirectoryFullPath;
    // ... constructor ...

    [KernelFunction, Description("Writes the given content to a specified file...")]
    public async Task<string> WriteFileAsync(
        [Description("The full path to the file to write.")] string filePath,
        [Description("The content to write to the file.")] string content)
    {
        if (!IsPathAllowed(filePath)) // Security Check
        {
            return "Error: Access to the specified path is denied.";
        }
        // ... ensure directory exists ...
        await File.WriteAllTextAsync(filePath, content);
        return $"Successfully wrote content to '{filePath}'.";
    }
    // ... ReadFileAsync, IsPathAllowed ...
}
```

Description attributes help the AI understand what the function does and what parameters mean.

Registering Tools

(Code snippet from Tools.cs)

```
public ToolFactory(ILoggerFactory loggerFactory, Settings settings)
{
    // ... logger setup ...
    var availableTools = settings.GetSettings<List<Settings.ToolSettings>>("tools");
    foreach (var tool in availableTools)
    {
        // ... check if already registered ...
        switch (tool.Name)
        {
            case "FileSystem":
                // ... check for 'basepath' parameter in settings ... [8]
                Register("FileSystem", () => KernelPluginFactory.CreateFromObject(
                    new FileSystemPlugin(tool.Parameters["basepath"])));
                break;
            // ... other tools ...
        }
    }
}
// ... GetTool, Register methods ...
```

Creates a KernelPlugin using KernelPluginFactory.CreateFromObject()

Initializing Agents

(Code snippet from Agents.cs)

```
public void InitializeAgents(..., ToolFactory toolFactory, ...)
{
    // ... clear agents, get settings ...
    var agentSettings = settings.GetSettings<List<Settings.AgentSettings>>("agents");
    foreach (var agentSetting in agentSettings)
    {
        var kernel = KernelFactory.CreateKernel(...); // Create kernel for this agent
        if (agentSetting.Tools.Count > 0)
        {
            foreach (var toolName in agentSetting.Tools)
            {
                var tool = toolFactory.GetTool(toolName); // Get plugin from factory
                if (tool != null) { kernel.Plugins.Add(tool); } // Add plugin to kernel
            }
        }
        var configuredAgent = new ChatCompletionAgent {
            Name = agentSetting.Name,
            Description = agentSetting.Description,
            Instructions = agentSetting.Instructions,
            Kernel = kernel, // Assign kernel with plugins
            // ... Arguments/ExecutionSettings for tool calling ... [2]
        };
        _availableAgents.Add(configuredAgent);
    }
}
```

Creates a dedicated Kernel for each agent, including tools registration

Setting up the chat

(Code snippet from Scenarios.cs)

```
private AgentGroupChat chat = new();
// ... inside Initialize method ...
var scenarioSettings = ... // Get specific scenario settings
var scenarioKernel = KernelFactory.CreateKernel(...); // Kernel for orchestration
var activeAgents = ... // Get agent instances for this scenario

var terminateFunction = AgentGroupChat.CreatePromptFunctionForStrategy(scenarioSettings.TerminationPrompt);
var selectionFunction = AgentGroupChat.CreatePromptFunctionForStrategy(scenarioSettings.SelectionPrompt);

chat.ExecutionSettings = new AgentGroupChatSettings {
    TerminationStrategy = new KernelFunctionTerminationStrategy(terminateFunction, scenarioKernel) { ... },
    SelectionStrategy = new KernelFunctionSelectionStrategy(selectionFunction, scenarioKernel) { ... }
};

foreach (var agent in activeAgents) { chat.AddAgent(agent); }
```

Creates an AgentGroupChat instance and a dedicated Kernel for running orchestration prompts

Orchestration Strategies

(Code snippet from Scenarios.cs)

```
chat.ExecutionSettings = new AgentGroupChatSettings
{
    TerminationStrategy = new KernelFunctionTerminationStrategy(terminateFunction, scenarioKernel)
    {
        Agents = agents.Where(a => scenarioSettings.TerminationAgents.Contains(a.Name)).ToList(),
        HistoryVariableName = "history",
        ResultParser = (result) => result.GetValue<string>()?.Contains(scenarioSettings.TerminationSuccess, StringComparison.Inv
    },
    SelectionStrategy = new KernelFunctionSelectionStrategy(selectionFunction, scenarioKernel)
    {
        AgentsVariableName = "agents",
        HistoryVariableName = "history"
    }
};
```

Strategies help to determine the next agent or when it's time to stop

Running the Chat

(Code snippet from Scenarios.cs)

```
public async Task Execute(string prompt)
{
    // Add initial user message
    chat.AddChatMessage(new ChatMessageContent(AuthorRole.User, prompt));

    // Invoke the chat
    await foreach (var content in chat.InvokeAsync())
    {
        _logger.LogInformation($"# {content.Role} - {content.AuthorName ?? "*"}: '{content.Content}'");
        // Potentially add logic here based on message content/author
    }

    _logger.LogInformation($"# IS COMPLETE: {chat.IsComplete}");
}
```

The loop processes each turn using the orchestrator strategies

KEY TAKEAWAYS: TAMING THE CATS IS POSSIBLE!

Taming the AI Cats

Orchestration is Possible

Specialization Works

Prompts are Powerful

Configuration is Key

Tools enable Actions

Growing your AI Team

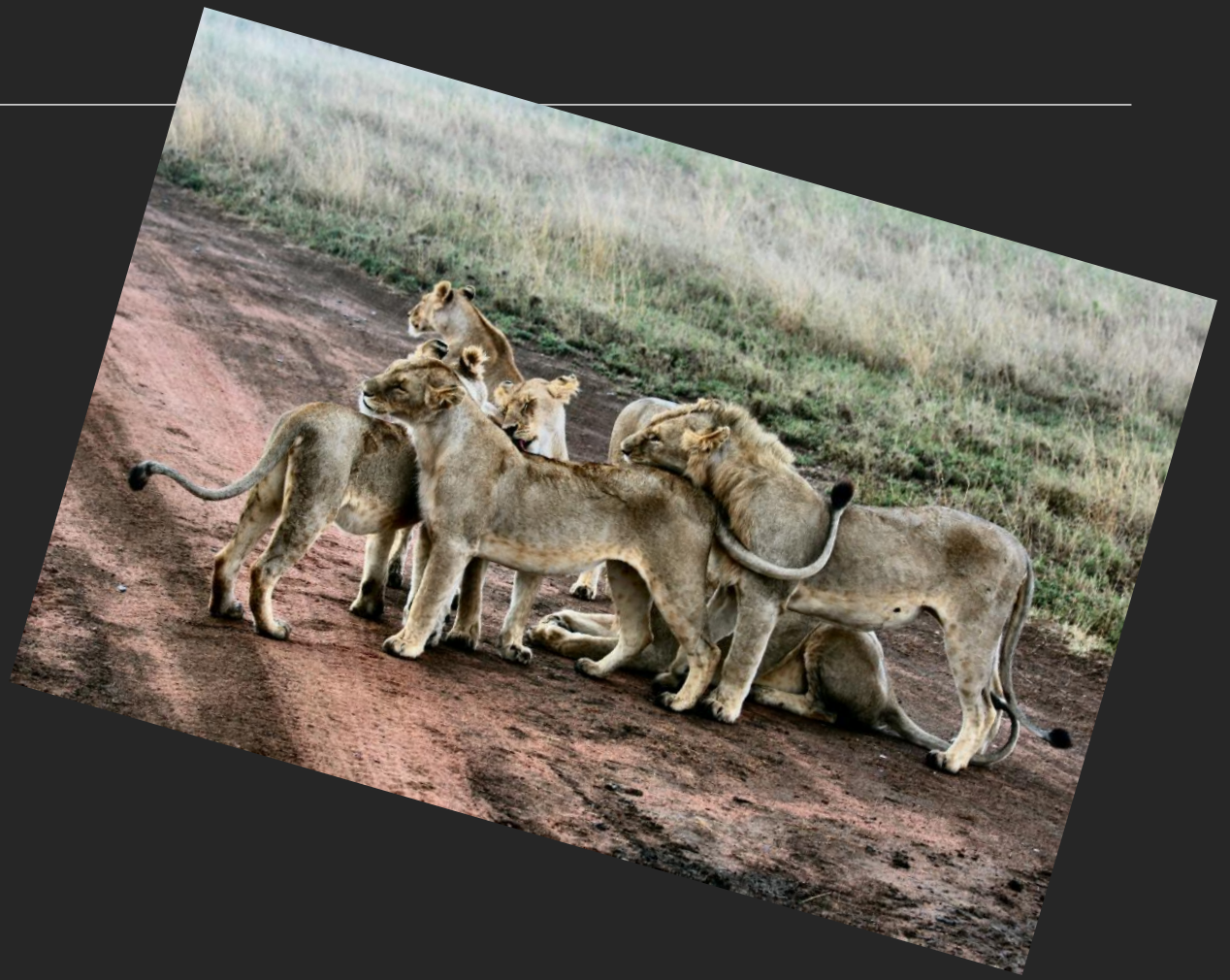
Adding / Modifying Agents

Adding / Modifying Tools

Additional Scenarios

Change Workflow Logic

New Integrations



Best Practices and Considerations

Prompt Engineering is Vital

Robust Configuration

Error Handling & State

Cost & Performance

Security

Observability

SO LONG, AND THANKS FOR ALL THE FISH! LINKS AND Q&A

Links

Code & Documentation: https://github.com/mplogas/sk-multiagent

“Debugging on Live” Podcast: https://www.youtube.com/@debuggingonlive

“Developer’s Cantina” Blog: https://www.developerscantina.com/

Semantic Kernel: https://learn.microsoft.com/en-us/semantic-kernel/overview/

Global AI Community: https://globalai.community/

Questions?


@mplogas@mastodon.social


mplogas


mplogas

Thank you!


@mplogas@mastodon.social


mplogas


mplogas