21/11 – session

LLM-Based Applications

Building applications powered by Large Language Models (LLMs) present unique challenges that don't exist in traditional software development. Many organizations face obstacles in bringing LLM-based solutions to production, often due to a lack of specialized knowledge and tools, leading to project failures.

In upcoming Microsoft Reactor sessions (21-Nov) and the next session, we will develop a full-blown RAG (Retrieval Augmented Generation) application. We will share with you all the techniques, frameworks, and best practices to build end to end Gen AI applications.

Participants will gain the necessary skills to design, develop, and deploy LLM-based applications effectively. Through in-depth sessions, we will explore the nuances of LLM system development, introduce cutting-edge tools and frameworks, and share best practices for evaluatrion and continuous integration. Join us to navigate the complexities of LLM applications and drive innovation in your organization.

Key Questions Addressed:

- **What distinguishes the development of LLM-based systems from conventional software systems?**
- **Which tools and frameworks can accelerate LLM application development?**
- **What are the essential components of LLM-based applications?**
- **How should LLM-based applications be tested?**
- **Is it feasible to apply existing CI/CD methodologies and tools to LLM-based systems?**
- **What strategies facilitate the seamless integration of LLM-based applications within an organization?**

LLM-Augmented Applications

General Note

> I don’t like the term "Gen AI applications" because it implies that Gen AI applications are a separate category of systems. They are not. There is no need to invent "special implementations for Gen AI."
> Very often, I see people advertise "AI Powered something," but everything should be AI-powered these days.
> AI is born to augment and elevate any solution to a new level.
> Gen AI should be considered an integral part of every software system. Don’t focus on developing Gen AI bots; instead, think about where AI can improve and elevate your solutions to the next level, provide new features, whether in existing and new products, software development, or maintenance and support.
> Embrace AI everywhere.

Types of LLM applications

In my opinion, there are two types of LLM applications: **LLM-Based** and **LLM-Augmented**.
LLM-Based applications are those where core operations fully rely on the LLM. In contrast, LLM-Augmented applications are those where the LLM enhances and enriches certain aspects but is not necessarily central to the application’s primary functions.
For the sake of simplicity, I’ll use the term LLM Applications in this post, except in cases where there is an explicit difference that must be noted.

Paradigme switch

Incorporate AI into your products and avoid developing standalone, brand-new AI-centric applications just because they use AI. The implication of this approach is that you use mostly the same development methodologies and principles. Your development cycle remains the same as before, but you augment it with the superpower of AI.
For example, consider the development of a new feature, which consists of the following phases:

- Requirements collection and analysis
- Define feature objectives and scope
- Design
- Implementation and testing
- Deployment
- Maintenance

In every step, you can incorporate AI. Acting this way will bring many new ideas and provide your company with a significant competitive advantage and speed.
The management, product, developers, operations, and support teams should be educated to act this way. This paradigm shift should be supported at all levels in your company.
One of the most powerful tools to empower and motivate people in this matter is AI workshops followed by 1-2 day hackathons. Employees bring their ideas and implement them during the hackathon.

With great power comes great responsibility

Of course, AI cannot be just "dropped" into organizations for free and uncontrolled usage, as this will bring risks such as new security vulnerabilities, high costs, and product quality degradation. Every organization entering the AI world should address the following questions regarding LLM Applications:

1. What are the architectural approaches, patterns and techniques?
2. Should we provide a cross-organization AI platform to share common functionalities, like models, LLM agents, and so on?
3. How do we secure LLM Applications?
4. Which Gen AI frameworks should we use to speed up development?
5. How do we evaluate the quality of LLM Applications?
6. How do we monitor LLM Applications?
7. How do we ensure full cost visibility and control?
8. How do we prepare data for LLM Applications?

You can see that many questions from this list are the same as for standard software projects, but there are still some new aspects we're going to cover here.

Architecture patterns

When incorporating LLMs directly into your code without any refactoring, you leverage the existing architecture. However, you should consider the latencies that LLMs introduce into your application flow. Addressing latency SLAs might require partial redesign and refactoring, or you might decide to use small and fast language models (SMLs) that are quick enough and provide sufficient quality to adhere to performance SLAs. Even in this scenario, you must address topics such as security, evaluation, monitoring, and LLM service availability.

Monolith vs. Microservices

If you are building a brand new LLM application, you should apply the same system design principles as in standard software architectures. The current trend in LLM applications is moving towards a microservices approach, known in the LLM applications world as the multi-agent pattern.

Think microservices: define unique responsibilities for each microservice, design APIs and data sources, and select and define communication mechanisms between those services. Sounds familiar? In this context, a microservice powered by an LLM or SML is called an Agent. An Agent is usually backed by an LLM/SLM and can have a set of “tools” that the model can decide to apply to serve an incoming request. These tools can be internal/external APIs, web search APIs, other services, or agents that bring relevant context to the model.

Of course, you can add your additional business logic in an agent. As in the previous use case, you must address security, evaluation, monitoring, and LLM service availability.

Platform or not Platform

Should organizations provide an AI Platform for easier adoption? In most cases, an AI Platform can facilitate better sharing and reuse of common AI components like Agents, Vector DBs, and the deployments of base LLMs and fine-tuned models. An AI Platform can enforce common security policies and guidelines. Finally, an AI Platform can provide centralized means for cost control.

I’m not in favor of using large, opinionated AI Platforms, but prefer lightweight platforms with minimal dependency. An example of such a platform is Azure AI Studio, which provides the most basic tools for LLM applications. Specifically, it offers:

- A model catalog with over 1800 models (at the time of writing this post)
- A model deployment platform with managed and serverless options
- LLM application tracing and evaluation
- Budget management and cost controls
- A lightweight LLM application development framework, Prompt Flow
- Playground

With Azure AI Studio, you are free to use other third-party LLM frameworks. You have all the tools you need to focus on your business logic rather than maintaining infrastructure.

Gen AI frameworks

https://www.restack.io/p/semantic-kernel-answer-vs-langchain-vs-llamaindex-cat-ai

https://learn.microsoft.com/en-us/semantic-kernel/concepts/plugins/?pivots=programming-language-python

1. LangChain
2. Semantic Kernel

LangChain

Overview:
LangChain is an open-source development framework designed to facilitate the creation of applications powered by Large Language Models (LLMs). It provides a robust and flexible environment for integrating various components to perform complex tasks.

Key Concepts:
Chains: These are structured sequences of operations that integrate various components, such as LLMs, vector databases, APIs, and data preprocessing tools, to perform complex tasks.

Supported Languages:

- Python
- JavaScript
- TypeScript

Use Cases:
LangChain is ideal for projects requiring high flexibility and rapid prototyping. It is suitable for building applications from scratch with seamless integration of components like vector databases, LLMs, and memory modules.

Semantic Kernel

Overview:
Semantic Kernel is an open-source development framework that combines prompt engineering with existing APIs to perform actions, serving as middleware between AI models and function calls.

Key Concepts:
Plugins: These encapsulate existing APIs into collections that can be utilized by AI models, facilitating function calling.

Function Calling: This leverages LLM capabilities to invoke specific functions based on user requests, with Semantic Kernel translating these requests into function calls and returning results to the model.
Integration: Enables the connection of existing codebases to AI models, enhancing the existing functionality without extensive redevelopment.

Supported Languages:

- C#
- Java

- Python

Use Cases:
Semantic Kernel is suited for enterprise applications requiring higher control over application flow. It is effective in integrating existing code with AI models, allowing for the augmentation of current systems with AI capabilities.

Agentic Frameworks

1. LangGraph
2. AutoGen
3. Crewai

Evaluation LLM Applications

Evaluation Data Set

The first step in designing and planning your evaluation strategy is preparing evaluation data sets. Creating these data sets is a critical part of LLM evaluation. The evaluation data set depends on the scenario your application handles. If you build a multi-agent system, prepare an evaluation data set for every agent along with an evaluation data set for the entire system.
You can develop, test, and evaluate every agent independently. Once all agents are evaluated, you should evaluate the whole system with a dedicated evaluation data set. Consider continuous data set updates and enrichment with anonymized production data.
[Run evalution as part of CI/CD]

Example of data set for simple QnA application:

Question	**Expected Answer**
What is the capital of France?	Paris

Who wrote the play “Romeo and Juliet”?	William Shakespeare
What is the boiling point of water at sea level in degrees Celsius?	100 degrees Celsius
Which planet is known as the Red Planet?	Mars
What is the chemical symbol for gold?	Au
Who is the author of the “Harry Potter” series?	J.K. Rowling

Evaluation metrics

For each scenario (agent), define key evaluation metrics to measure. For each metric, establish a minimum threshold that the agent must meet to be allowed deployment to production.
Any metric value that falls below this threshold should be considered an application build failure.
Some metrics require LLMs for evaluation, while others do not.
You can design your own evaluation metrics, or for specific scenarios like conversational applications, you can use metrics designed and implemented by Microsoft.

For instance, Microsoft provides the following metrics for conversational scenarios:

- **Groundedness:** Assesses how well the model’s responses align with the provided source information.
- **Relevance:** Measures the extent to which the generated responses are pertinent to the given questions.
- **Coherence:** Evaluates the logical flow and clarity of the generated text.
- **Fluency:** Determines the grammatical accuracy and naturalness of the language used in the responses.
- **Similarity:** Quantifies how closely the generated text matches a reference or expected output.

Azure AI Studio provides tools for both manual and automatic evaluations. Additionally, it offers a prompts library for the metrics described above, which you can use when building your evaluation flows.

DEMO Azure AI Studio Demo with manual evaluation
DEMO Azure AI Studio Demo with prompt library

Automatic evaluation

For automatic evaluations Microsoft designed and implemented “AI-assisted evaluaitons”
AI-assisted evaluations framework use LLM for evaluations.
It provides evaluation metrics in 2 main categories:

1. Performance and quality
2. Risk and safety

Azure evaluation SDK

You can use this SDK to develop automatic evaluations.

Usage Scenarios:

Query and Response: Evaluate applications that generate responses to user queries.

Retrieval Augmented Generation: Assess models that generate responses using retrieved information from provided documents.

Supported Evaluation Metrics Categories:

Performance and Quality (AI-assisted): Metrics such as groundedness, relevance, coherence, fluency, similarity, and retrieval.

Performance and Quality (NLP): Metrics including F1 score, ROUGE, GLEU, BLEU, and METEOR.

Risk and Safety (AI-assisted): Metrics like violence, sexual content, self-harm, hate/unfairness, jailbreak indirect attack, and protected material.

Composite: Combined evaluators like QAEvaluator and ContentSafetyEvaluator.

Usage example:

```
pip install azure-ai-evaluation
```

```
from azure.ai.evaluation import RelevanceEvaluator
relevance_eval = RelevanceEvaluator(model_config)

relevance_score = relevance_eval(
   response="The Alpine Explorer Tent is the most waterproof.",
   context="From our product list, the Alpine Explorer Tent is the most waterproof. The Adventure Dining Table has higher weight.",
   query="Which tent is the most waterproof?"
)
print(relevance_score)
```

```
{'relevance.gpt_relevance': 5.0}
```

Data preparation and retrieval in RAG

The retrieval component in RAG (Retrieval Augmented Generation) applications plays a critical role. If it misses relevant texts or retrieves irrelevant ones, the overall performance of the RAG application will be poor. In this section, we discuss and provide best practices and recommendations for achieving high retrieval performance.

Data Types

Input data for RAG applications is often multi-modal and multi-lingual. It consists of texts, images, videos, and audio. Very often, even textual data contains complex structures like tables, diagrams, and flows.

Embedding

The multi-modal data is embedded, and the embedding vectors are saved in a search engine to allow vector (semantic) search. Some search engines, like Azure AI Search, support both classic keyword and vector search. In Azure AI Search, you can apply both types of search in the same query, known as hybrid search, which can improve search recall.

Reranking

To improve retrieval precision, a good technique to consider is reranking. At its core, reranking usually involves small language models that take a query and the retrieved text from the search, then provide a relevancy score indicating how relevant the answer is to the input query. Azure AI Search has built-in reranker models, which you can use along with vector search to improve search precision.

Filters

Along with embedding vectors and texts, you can keep additional attributes that allow you to filter relevant records and then apply a vector search. This technique can further improve search precision. AI Search supports filters as part of its data structure called an index.

Data preparation

https://learn.microsoft.com/en-us/azure/search/vector-search-how-to-chunk-documents

One of the most common questions asked is how to chunk input text documents to achieve the highest precision and recall. You can chunk documents using one of the following approaches:

a. **Fixed-size chunks:** The recommended chunk size is approximately 400 words with a 10-15% overlap between adjacent chunks.
b. **Variable-sized chunks based on content:** Chunk by pages, paragraphs, and sub-paragraphs.

c. **Advanced chunking techniques:** Divide the document into small chunks, such as sentences. Embed the chunks and merge adjacent chunks with very high semantic similarity.

Both LangChain and Semantic Kernel provide chunking functionality, allowing you to choose the best approach for your specific use case.

https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/
https://learn.microsoft.com/en-us/dotnet/api/microsoft.semantickernel.text.textchunker

OCR

Often, documents contain graphical objects like diagrams, charts, flows, and images with text. To enhance search recall further, you can use Azure Document Intelligence service to retrieve the text from these graphical objects. Another option is to use small and cost-effective models like gpt4o-mini to extract context from graphical objects and process it as regular text.

Non-Textual Data
To support multi-modal search, you should be able to embed non-textual data and save the embedding vectors in a vector database. In Azure, you can use Azure AI Vision Multimodal Embeddings to embed images. This service generates vector representations for images, allowing you to embed both texts and images and store the vectors in Azure AI Search. This enables you to run multi-modal searches effectively.

MS Reactor. Session 28/11

Hands on workshop

Workshop at Microsoft TLV Reactor on Thursday, 28/11.

In this hands on workshop, we will develop a complete RAG LLM Agent from scratch, including advanced quality evaluation and monitoring.

Prerequisites

- Access to an Azure AI Search service instance.
- Access to Azure AI Studio or Azure Open AI with the following deployed models: gpt-4o, gpt-4o-mini, text-embedding-ada-002.
- PDF documents stored in a storage account for building RAG.
- VS Code.

Required Permissions (RBACs)

- **Azure AI Search**: Search Service Contributor, Search Index Data Contributor
- **Azure Open AI**: Cognitive Services Contributor/User
- **Azure AI Studio**: Contributor or Azure AI Developer (AI Studio project level)
- **Azure Storage Account**: Storage Blob Contributor/Reader

Development environment

- VS Code
- Prompt Flow VS Code extension
- GitHub
- Python version: 3.11.10
    - In mac you can install Python bt running: brew install python@3.10
- requirements.txt is in the project root folder (llmops) contains all dependencies

Setting up the project

1. Create a project folder and cd to it
2. Clone the project : git clone git@github.com:vladfeigin/azure-dai-demos.git
3. Open VS Code and select the project folder
4. Create a Python virtual environment:

   In VS Code, click on the ⚙ icon, select “Command Palette”.

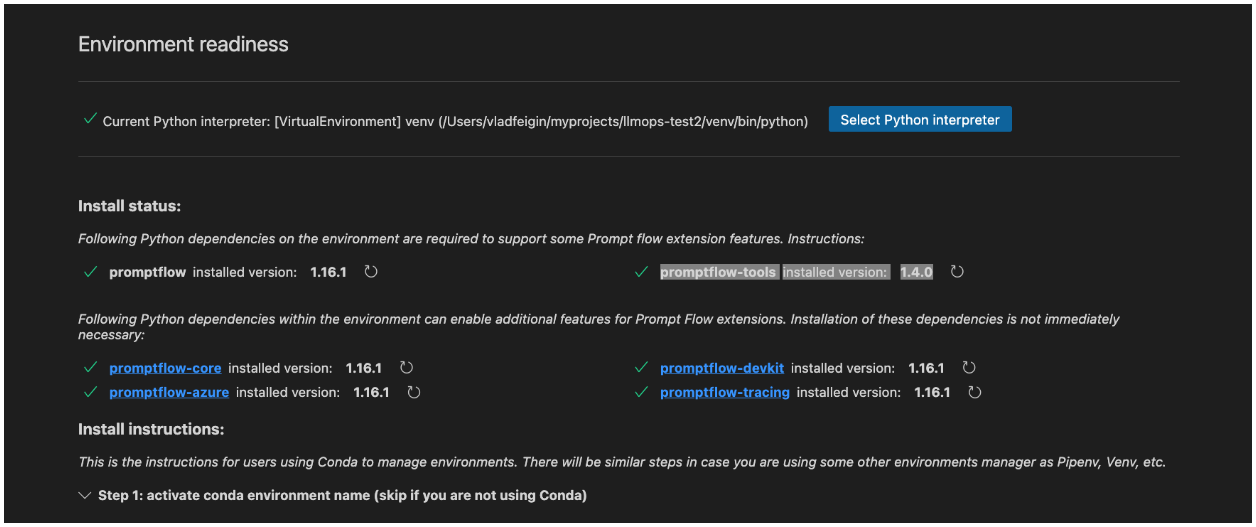
In the dialog box type “Python:Create Environment”.
Select “Create a .venv ...”
Enter interpreter path.. Python installation path.
Check box: azure-dai-demos/llmops/requirements.txt
Click ok.

- If this option did not work, alternatively run the following commands in the project folder:

```
python3.10 -m venv venv
(Windows: venv\Scripts\activate)
source ./venv/bin/activate
pip install -r requirements.txt
```

Make sure you are in ./llmops directory!

5. Install Prompt Flow VS Code extension
6. Click on Prompt Flow icon and select Install Dependencies.
   a. Select Python interpreter with a correct Python version
   b. Install missing Promp Flow dependencies



Environment readiness

✓ Current Python interpreter: [VirtualEnvironment] venv (/Users/vladfeigin/myprojects/llmops-test2/venv/bin/python)

Install status:

Following Python dependencies on the environment are required to support some Prompt flow extension features. Instructions:

✓ **promptflow** installed version: **1.16.1** ↻   ✓ **promptflow-tools** installed version: **1.4.0** ↻

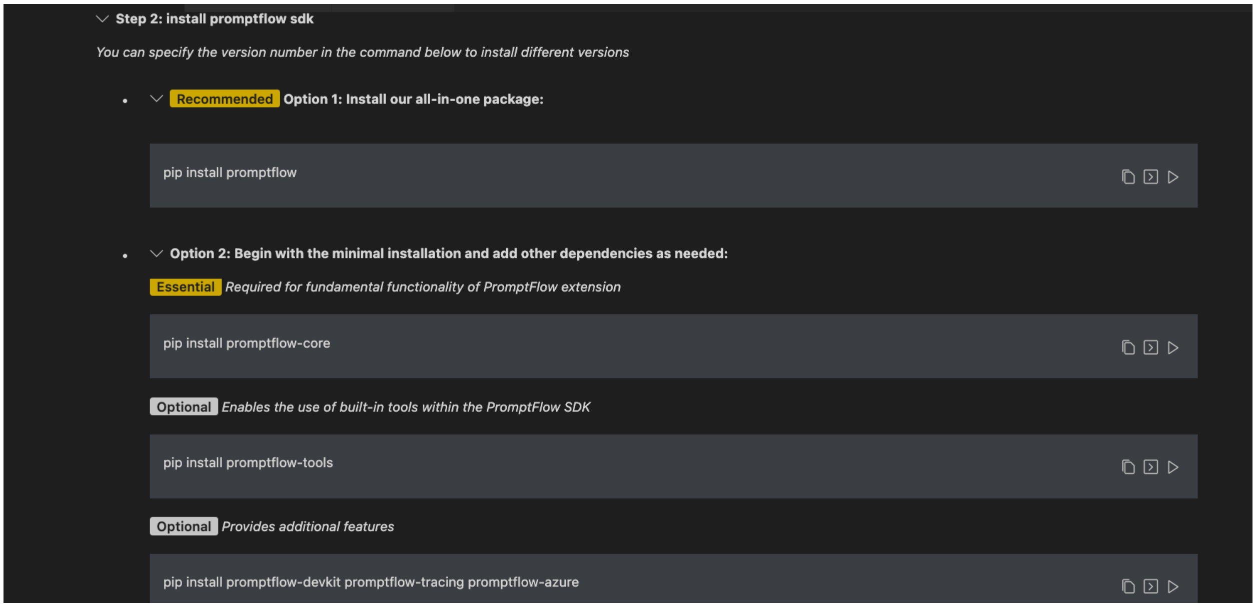
Following Python dependencies within the environment can enable additional features for Prompt Flow extensions. Installation of these dependencies is not immediately necessary:

✓ **promptflow-core** installed version: **1.16.1** ↻   ✓ **promptflow-devkit** installed version: **1.16.1** ↻
✓ **promptflow-azure** installed version: **1.16.1** ↻   ✓ **promptflow-tracing** installed version: **1.16.1** ↻

Install instructions:

This is the instructions for users using Conda to manage environments. There will be similar steps in case you are using some other environments manager as Pipenv, Venv, etc.

Step 1: activate conda environment name (skip if you are not using Conda)



Step 2: install promptflow sdk

You can specify the version number in the command below to install different versions

- **Recommended** **Option 1: Install our all-in-one package:**

```
pip install promptflow
```

- **Option 2: Begin with the minimal installation and add other dependencies as needed:**

Essential *Required for fundamental functionality of PromptFlow extension*

```
pip install promptflow-core
```

Optional *Enables the use of built-in tools within the PromptFlow SDK*

```
pip install promptflow-tools
```

Optional *Provides additional features*

```
pip install promptflow-devkit promptflow-tracing promptflow-azure
```

7. In Azure AI Studio/Foundry, create a project and deploy the following models:
   a. gpt-4o,
   b. gpt-4o-mini,
   c. text-embedding-ada-002

Open file: ./rag/rag_agent_config.yaml. Change model_deployment_endpoint value to your deployment URL from Azure AI Studio.

8. Fill in the ./llmops/.env_template file with your environment values and rename it to .env

9. Test the project:
   a) Navigate to the project root folder: ` cd ./llmops`
   b) Execute:

```
pf service stop
pf service start
pf flow serve --source ./  --port 8080 --host localhost
```

Note: If you encounter an error:

["Failed to load python module from file '/private/var/folders/nm/syx206rd2cqb5m7jqvjd5z5r0000gn/T/tmpkd3cs2o9/runflow_local.py': (ModuleNotFoundError) No module named 'langchain_core'"]

when running ` pf flow serve --source ./  --port 8080 --host localhost ` , click the Prompt Flow icon, go to the "Environment readiness" section, and install all required dependencies. Ensure you have selected the correct Python interpreter version and are in the virtual environment.

This should open Web UI for chat, then insert a question and any session ID number.

1. Run in web browser: http://localhost:23337/v1.0/ui/traces/ - to see the traces.

Note that in order to generate Open Telemetry traces you just need put into your code:

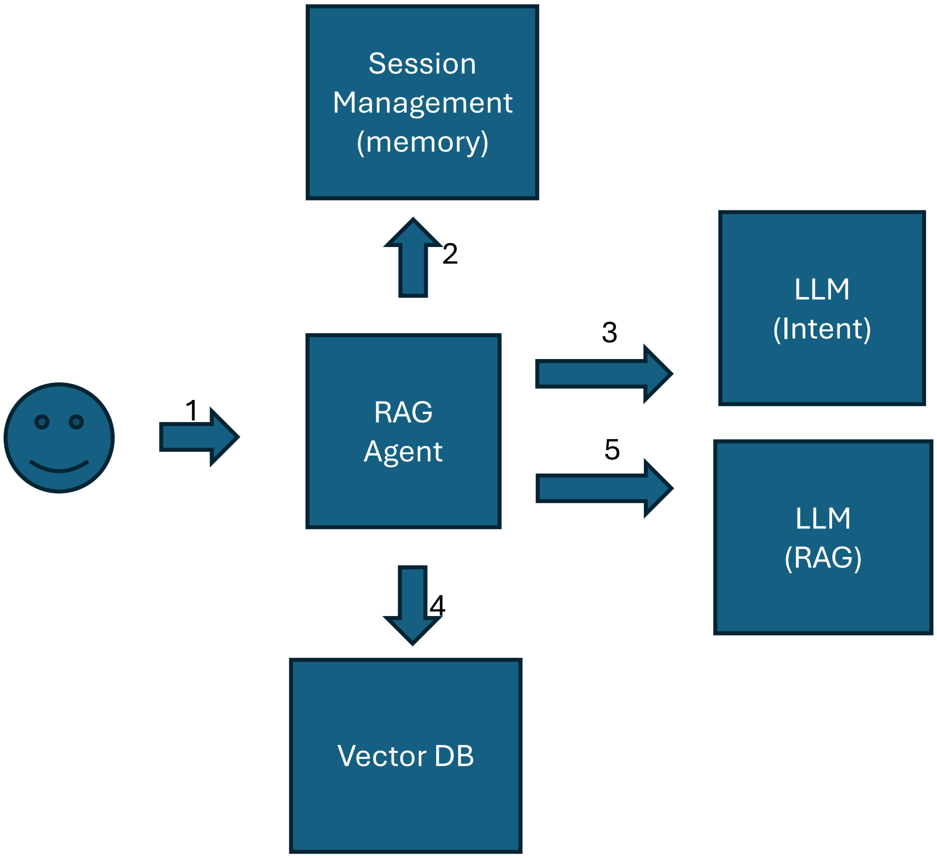
```python
from promptflow.tracing import start_trace
start_trace()
```

Explore the traces to understand the flow.

You are now ready for the workshop.

Creating an LLM-based RAG app in Azure from scratch.

Architecture



Flow

In this workshop we use the same LLM model for both RAG agent and Intent Agent but those two could be separated into different Agents.

- User calls the RAG Agent.
- Agent checks if there is a chat history by verifying if a session ID is presented in the request header.
- If it's not a new session, Agent retrieves the chat history from the "Session Management" service.
- Agent sends the history and the current question to the LLM model to identify the current user intent.
- Intent Agent identifies the user's intent and prepares the query for search.
- RAG Agent sends the query to a Vector DB.
- RAG Agent receives the context and, along with the chat history, sends it to the LLM model.

In this workshop, we use the same LLM model for both RAG Agent and Intent Agent, but these two could be separated into different Agents.

GenAI Framework

In this workshop we use LangChain framework
Main LangChain usage is in **rag** module (rag_main.py)
We user a several LangChain shougar methods:

a. *create_history_aware_retriever* – for intent identification
b. *create_stuff_document_chain* – for a regualar passing documents to the model
c. *create_retrieval_chain* - encapsulates above "a" and "b"
d. *RunnableWithMessageHistory* – runs *create_retrieval_chain* along with automatic chat history handling

Note that in "chat" method we send custom Tracing attributes like:

session_id, config_version (variant). Traces are collected in MS Fabric for further analysis.

Retrieval

As a Vector DB we use Azure AI Search

Intro AI Search

Main AI Search concepts.

1. Index
2. Indexer
3. Skills
4. Data Sources

Demo

- Prepare the documents for RAG and copy them to a container in Azure Storage Account. (storage account: genaiworkshopstorage1)
- Assign all required roles to AI Search Managed Identity: *Storage Blob Data Contributor, Storage Blob Data Reader.*
- Create an index in AI Search from the portal. Select the option to "Import and vectorize data" in the portal.
- Create reranker (Select option of "Semantic Query") in "Import and vectorize data" wizard.
- Zoom-in into index. Check the index and run a simple query in the Query Explorer. Index name: vector-llmops-workshop-index

  Show the different score (reranker score for example)
  Update index fields: uncheck text_vector "Retrievable option.
  Pay attention on the default fields being created by the wizard:
  chunk_id, parent_id, chunk, title, text_vector

Retrieval implementation

LangChain has built-in integration with Azure AI Search

- A LangChain Retriever interfaces with Vector DB.
  We create an

  ```
  langchain_community.vectorstores.azuresearch.AzureSearch
  ```

  which is abstraction on top of Azure AI Search.

  After that we create a Retriever this way:

  ```
  self._retriever = self._vector_search.as_retriever(search_type=search_type,
  k=top_k)
  ```

  This retriever later is used in the method:

  ```
  create_history_aware_retrieve
  ```

  A history-aware retriever utilizes chat history to formulate a query, it employs a model for refinement before executing the search.

- Explore LangChain support for AI Search, including options such as rerankers support, scoring profiles, custom fields, and more.
- The index can be created from scratch, but then the documents must be ingested into it.
- Alternatively, an existing index can be used (the one we’ve created from the portal)

Session Management

Session management is essential for RAG scenarios as it maintains the conversation history.
In this workshop, we use a simple in-memory Session Management solution. Specifically, we utilize the LangChain ChatMessageHistory class, which stores chat history as a list in memory keyed by a session ID.
For production flows, use persistent services like Azure Cache for Redis.

RAG

If chat history is available, it is retrieved from the session management service and sent to the model to identify user intent. The identified user intent is then used for AI Search, and the resulting documents are sent to the model to generate a final response.
See rag.rag_main.py module for more details.

Evaluation

Evaluation Data Set

Developing an evaluation data set is a crucial step in initiating LLM application quality evaluations.
For conversational flows the template for such data set could be like this:

```
“session_id”:{}, “question”: {} “answer”: {} ,”context”:{}, “chat_history”: {}
```

Evaluation Data Set example

```
{"session_id":"1", "question": "What's Microsoft Fabric?","answer": "Microsoft Fabric
is an end-to-end analytics and data platform designed for enterprises that require a
unified solution. It encompasses various services such as Data Engineering, Data
Factory, Data Science, Real-Time Analytics, Data Warehouse, and Databases. Fabric
integrates these components into a cohesive stack, simplifying analytics requirements
by offering a seamlessly integrated, user-friendly platform. Key features: Unified
data storage with OneLake, AI capabilities embedded within the platform, Centralized
data management and governance,SaaS model.","context": "Microsoft Fabric is an end-to-
end analytics and data platform designed for enterprises that require a unified
solution. It encompasses data movement, processing, ingestion, transformation, real-
time event routing, and report building. It offers a comprehensive suite of services
including Data Engineering, Data Factory, Data Science, Real-Time Analytics, Data
Warehouse, and Databases. With Fabric, you don't need to assemble different services
from multiple vendors. Instead, it offers a seamlessly integrated, user-friendly
platform that simplifies your analytics requirements. Operating on a Software as a
Service (SaaS) model, Fabric brings simplicity and integration to your solutions.",
“chat_history”: {}}
```

Evaluation metrics

In this workshop we use following evaluation metrics:

Groundedness: Assesses how well the model’s responses align with the provided source information.
Relevance: Measures the extent to which the generated responses are pertinent to the given questions.
Coherence: Evaluates the logical flow and clarity of the generated text.
Similarity: Quantifies how closely the generated text matches a reference or expected output.

AI-Assisted Evaluations

The evaluation code is located within the “evaluation” module. It utilizes the azure.ai.evaluation package.
Example of Relevance evaluation:

```
#pip install azure-ai-evaluation

from azure.ai.evaluation import RelevanceEvaluator
relevance_eval = RelevanceEvaluator(model_config)

relevance_score = relevance_eval(
   response="The Alpine Explorer Tent is the most waterproof.",
   context="From our product list, the Alpine Explorer Tent is the most waterproof.\
  The Adventure Dining Table has higher weight.",
   query="Which tent is the most waterproof?"
)
```

print(relevance_score)

{'relevance.gpt_relevance': 5.0}

Variants

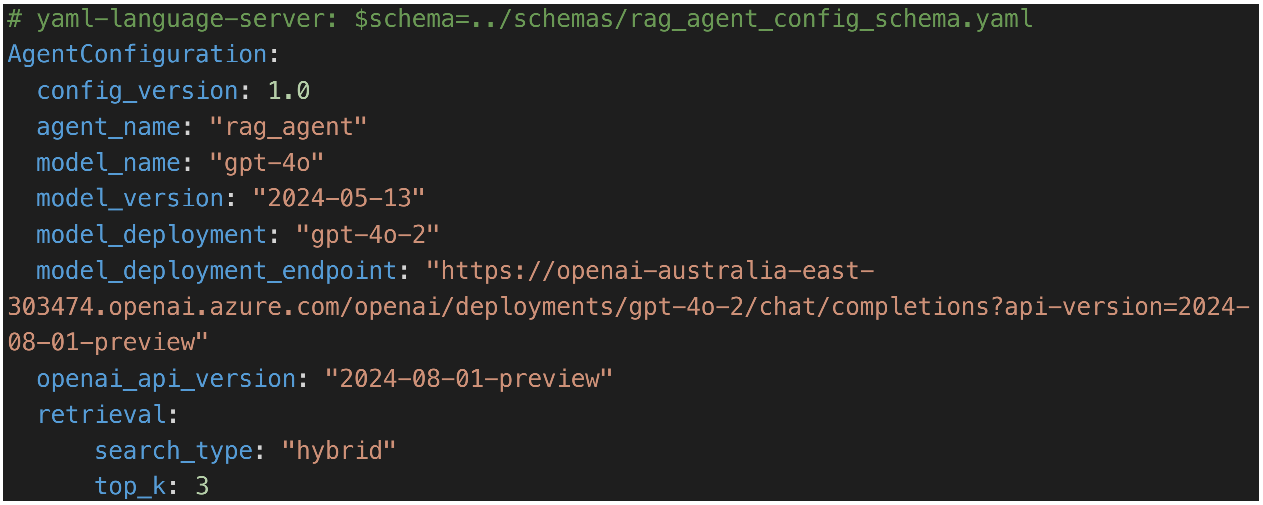
Imagine an application that employs several LLM Agents.
The performance of each agent and the entire application can be affected by a range of factors, such as:

- Model
- Model version
- Model parameters
- Prompts

So, how do we find the best configuration for optimal performance?
To tackle this, we use a **Variant**.
A Variant is a YAML file that outlines a specific revision of an Agent's configuration.
Each Agent evaluation run is tagged by the corresponding Variant.
We can create multiple variants for each Agent and automatically evaluate all of them and determine which one performs best.

The variant YAML file is in the same folder as the RAG Agent and loads automatically during initialization. The variant content is included in the traces, providing detailed agent configuration at the trace level. This information is collected in MS Fabric for comparing different variants.

Variant example:



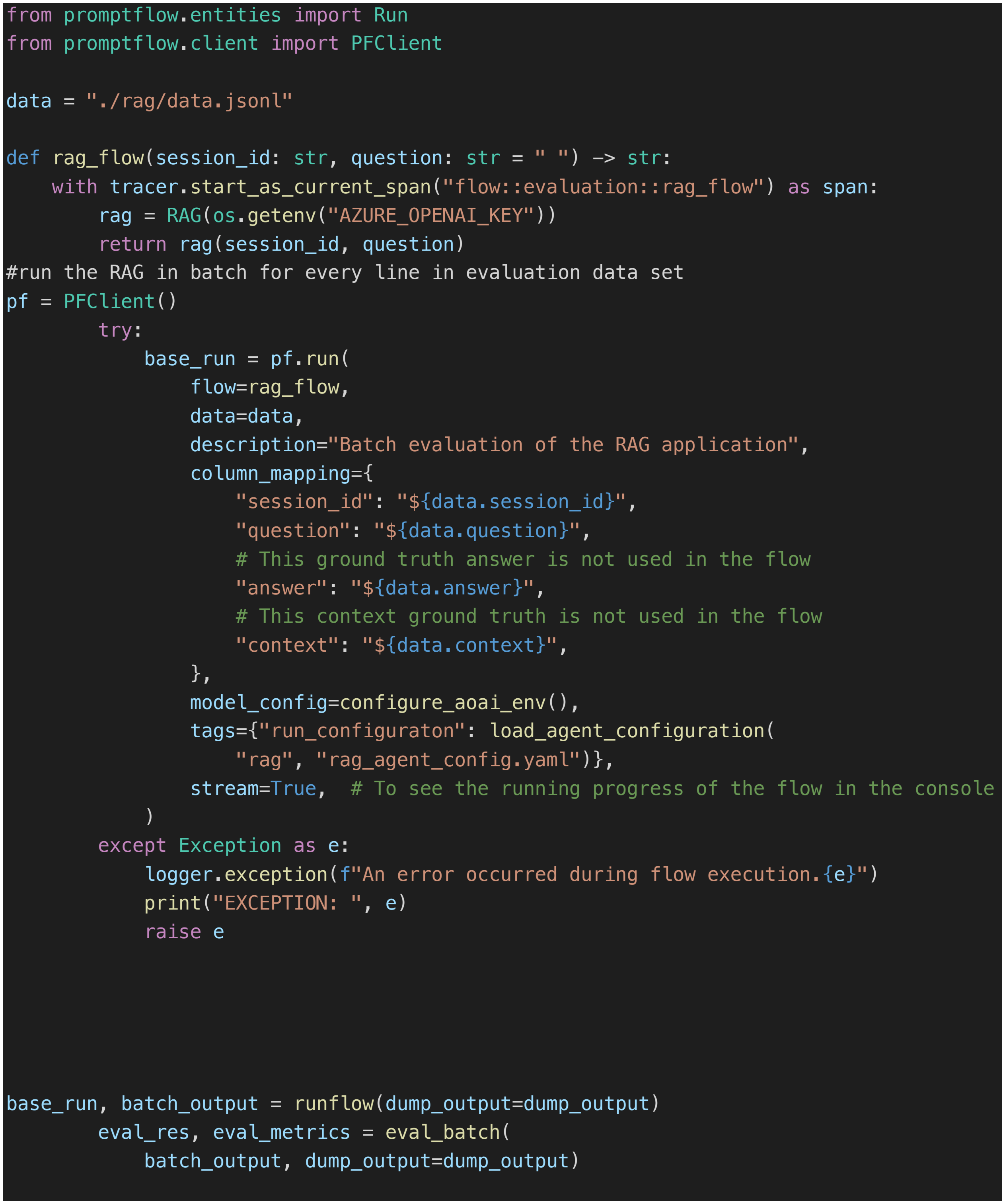
```yaml
# yaml-language-server: $schema=../schemas/rag_agent_config_schema.yaml
AgentConfiguration:
  config_version: 1.0
  agent_name: "rag_agent"
  model_name: "gpt-4o"
  model_version: "2024-05-13"
  model_deployment: "gpt-4o-2"
  model_deployment_endpoint: "https://openai-australia-east-303474.openai.azure.com/openai/deployments/gpt-4o-2/chat/completions?api-version=2024-08-01-preview"
  openai_api_version: "2024-08-01-preview"
  retrieval:
      search_type: "hybrid"
      top_k: 3
  model_parameters:
    temperature: 0.0
    seed: 42
  intent_system_prompt: >
        Your task is to extract the user’s intent by reformulating their latest question into a standalone query that is understandable without prior chat history.
        Analyze the conversation to identify any contextual references in the latest question, and incorporate necessary details to make it self-contained.
        Ensure the reformulated question preserves the original intent. Do not provide an answer; only return the reformulated question.
        For example, if the latest question is ‘What about its pricing?’ and the chat history discusses Microsoft Azure, reformulate it to ‘What is the pricing of Microsoft Azure?’

  chat_system_prompt: >
        You are a knowledgeable assistant specializing only in Microsoft technologies, including Azure, Microsoft Fabric, Microsoft 365, Dynamics 365, Power Platform.
        Your responses should be based exclusively on the context provided in the prompt; do not incorporate external knowledge.
        Address only questions related to Microsoft technologies. Deliver concise and clear answers, emphasizing the main points of the user’s query.
        If the provided context is insufficient to answer the question, request additional information.

        <context>
        {context}
        </context>"

  human_template: "question: {input}"
  #this parameter is deprecated in next version
  application_version: "1.0"   #- this parameter must be on application level configuration
  #this parameter is deprecated in next version
  application_name: "rag_llmops_workshop" #- this parameter must be on application level configuration
```

Evalution Implementation

We begin evaluation with prompt flow **batch run**, which takes an evaluation dataset file as input and returns model output for each line.

Then every line in the evaluation dataset, including model output from the batch run, is sent to the evaluation module.

For more details please see: runflow_local.py file.



```
from promptflow.entities import Run
from promptflow.client import PFClient

data = "./rag/data.jsonl"

def rag_flow(session_id: str, question: str = " ") -> str:
    with tracer.start_as_current_span("flow::evaluation::rag_flow") as span:
        rag = RAG(os.getenv("AZURE_OPENAI_KEY"))
        return rag(session_id, question)
#run the RAG in batch for every line in evaluation data set
pf = PFClient()
        try:
            base_run = pf.run(
                flow=rag_flow,
                data=data,
                description="Batch evaluation of the RAG application",
                column_mapping={
                    "session_id": "${data.session_id}",
                    "question": "${data.question}",
                    # This ground truth answer is not used in the flow
                    "answer": "${data.answer}",
                    # This context ground truth is not used in the flow
                    "context": "${data.context}",
                },
                model_config=configure_aoai_env(),
                tags={"run_configuraton": load_agent_configuration(
                    "rag", "rag_agent_config.yaml")},
                stream=True,  # To see the running progress of the flow in the console
            )
        except Exception as e:
            logger.exception(f"An error occurred during flow execution.{e}")
            print("EXCEPTION: ", e)
            raise e




base_run, batch_output = runflow(dump_output=dump_output)
        eval_res, eval_metrics = eval_batch(
            batch_output, dump_output=dump_output)
```

Tracing

Adding tracing is simple.
In order to generate Open Telemetry traces you just need put into your code:

```
from promptflow.tracing import start_trace
start_trace()
```

You can also add tracing spans, there are many examples in the code.

Observability