13 NOVEMBER 2024

slalom

Session 6: Chains and Agents

GenAI BootCamp

Northeast

Presentation contents

In this session

We'll cover

- Round robin
- Chaining and Agents Basics – LangChain
  - Schemas
  - Models
  - Prompts
  - Chains
  - Agents
- Chaining in Action Demo
- Challenge Project

You will

- Learn how & when to use chains specifically LangChain
- Understand the basics of Chaining
- Understand the basics of Autonomous Agents
- Understand Use Cases for both
- Witness chaining in action

Come prepared

- Bring your homework from last session and any questions
- Begin thinking about your Challenge Project

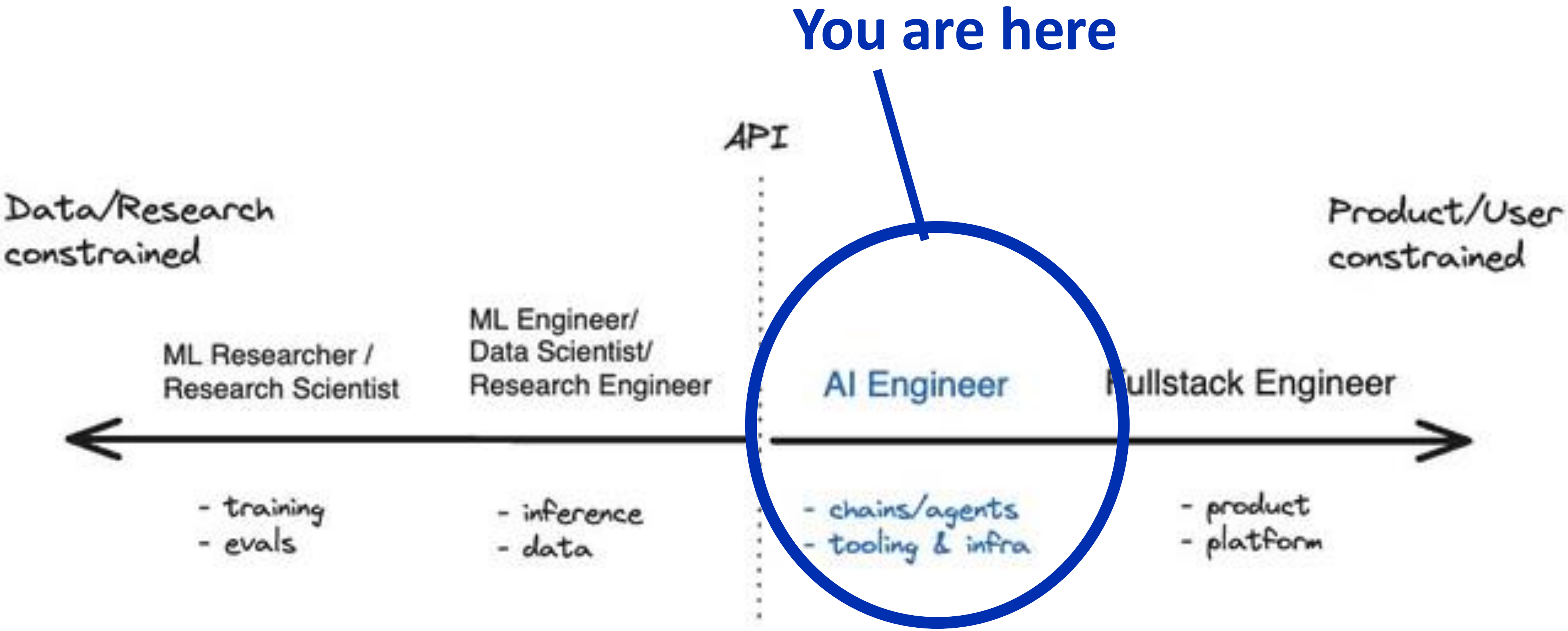
01 Round Robin – How'd your offline work go?

slalom

02 Chains and Agents

slalom

Fair warning



LLM Interaction Loop

Input
Prompts and context passed into the LLM

LLM
Used as a resource.

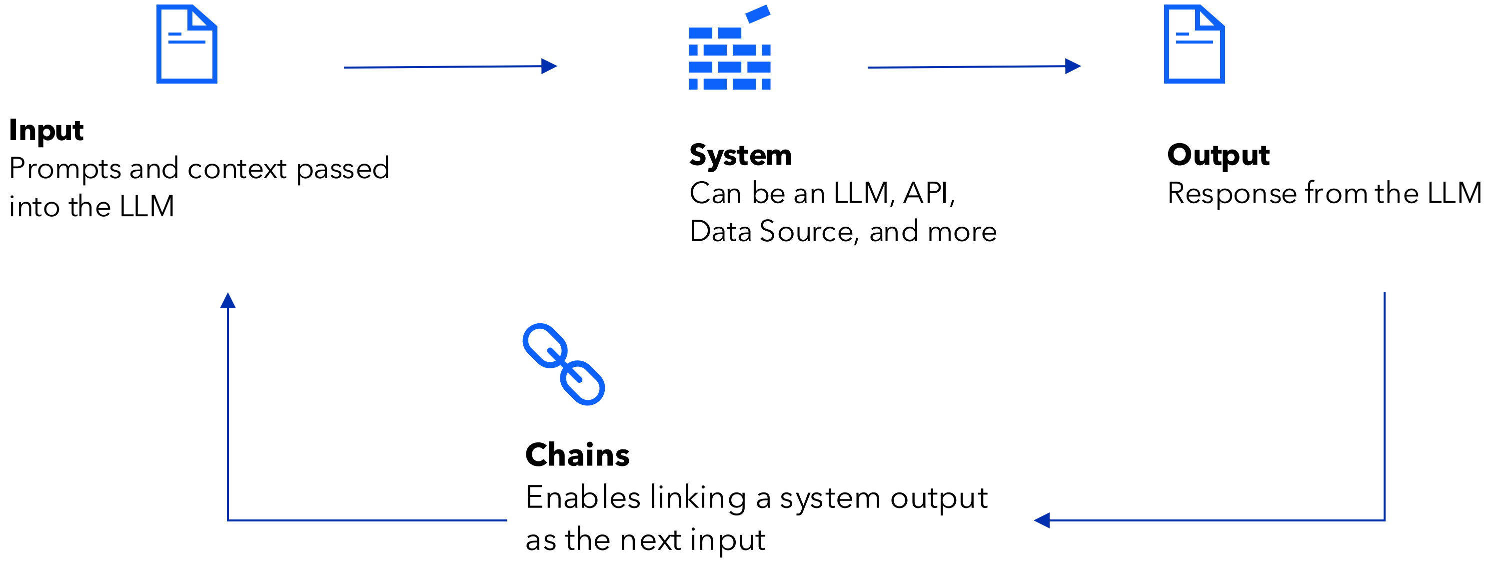
Output
Response from the LLM

Next Interaction
Manually look at the response and iterate on input until desired output reached

Highlights

- LLM interaction loop is usually a single interaction
- LLM doesn't have "memory", requires context to passed into with interactions
- May run into token limitations (e.g., 4096)

LLM Interaction Loop with Chains



Highlights

- Enables an abstraction layer to plug in different LLMs and systems
- Input Chain = Input -> System -> Output
- Different types of Chains available that can be linked together

Chain Examples

Using LangChain

LLM Chain

Calls different LLMs

Transformation Chain

Run a function on the input or output

Sequential Chain

Join chains together

Conversation Chain

Assumes a conversation is taking place

PAL Chain

Rewrites natural language into python code

SQL Database Chain

Rewrites natural language into a SQL Query

Bash Chain

Call bash commands

Request Chain

Requests an HTML page

API Chain

Call APIs

Chaining and Agents - LangChain

Chaining tools such as LangChain provide modular abstractions for the components necessary to work with models. LangChain also has collections of implementations for all these abstractions.

Schemas
Blueprint guiding the interpretation and interaction with data, mainly utilizing a "text in, text out" principle. This foundation lays out the ground rules before initiating the interaction.

Models
Primarily relies on three types of models: Large Language Models (LLMs), Chat Models, and Text Embedding Models, each playing a unique role in enhancing LangChain's versatility and strength.

Prompts
Functioning like the steering wheel guiding the model's direction based on the input questions or statements

Indexes / Memory

Efficiently organize and retrieve information to support easy and effective user interactions. Indexes in LangChain are crucial for processing and locating specific data amidst a massive amount of information. Memory persists context.

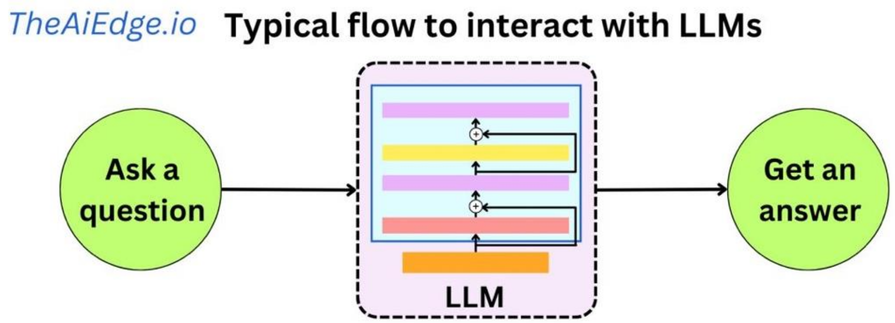
Chains
The system's master orchestrators, bringing together diverse elements to create responses from the language models that are meaningful and applicable, creating a system that is both unified and workable.

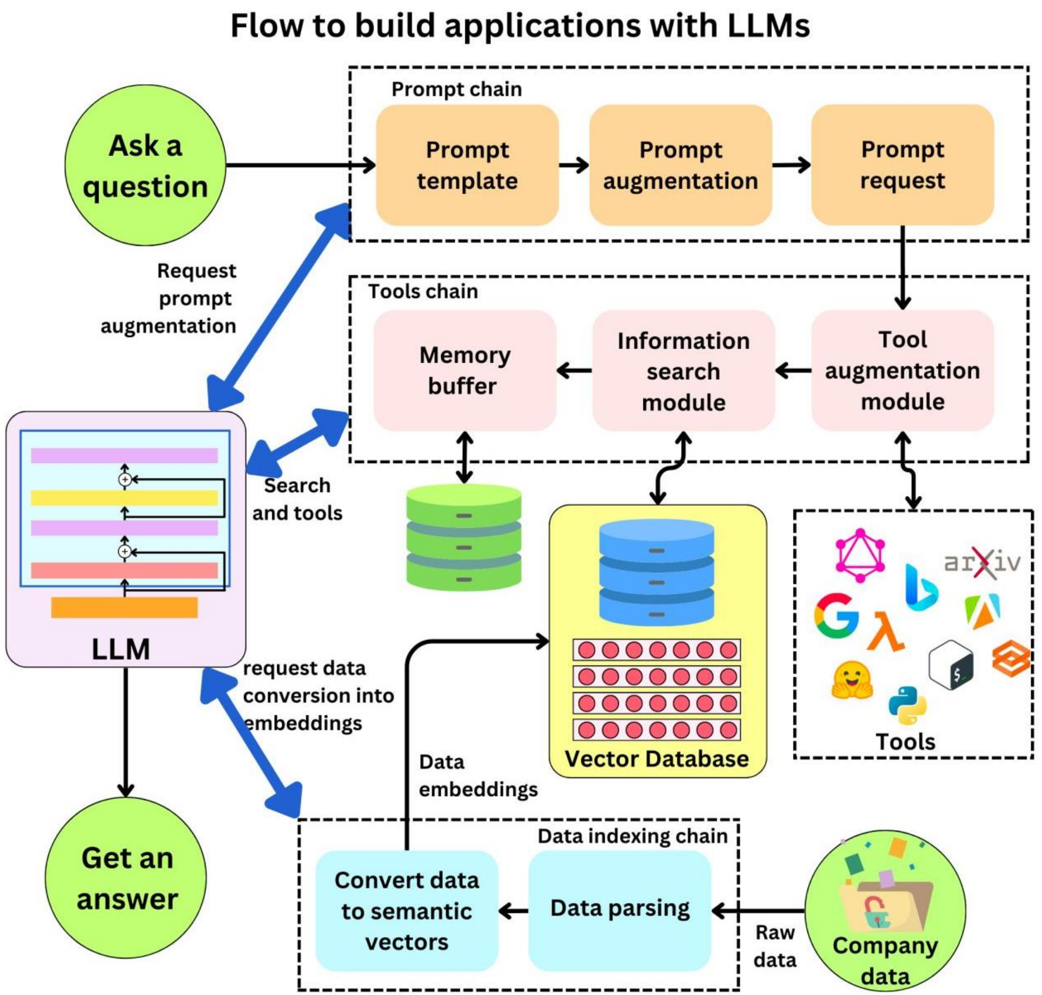
Agents
Action Agents perform like sprinters in a race, doing swift, precise actions, making them ideal for minor tasks. In contrast, Plan-and-Execute Agents are strategic and endurance-focused and excel at tackling difficult or long-term activities.

There are two main value props that chaining frameworks such as LangChain provides:

- *LangChain provides modular abstractions for the components necessary to work with language models.*
- *Use-Case Specific Chains: Chains can be thought of as assembling these components in particular ways in order to best accomplish a particular use case.*

Building Real Applications with LLMs





Agents

An agent uses an LLM to ***choose*** a sequence of actions to take, building a chain ***dynamically***.

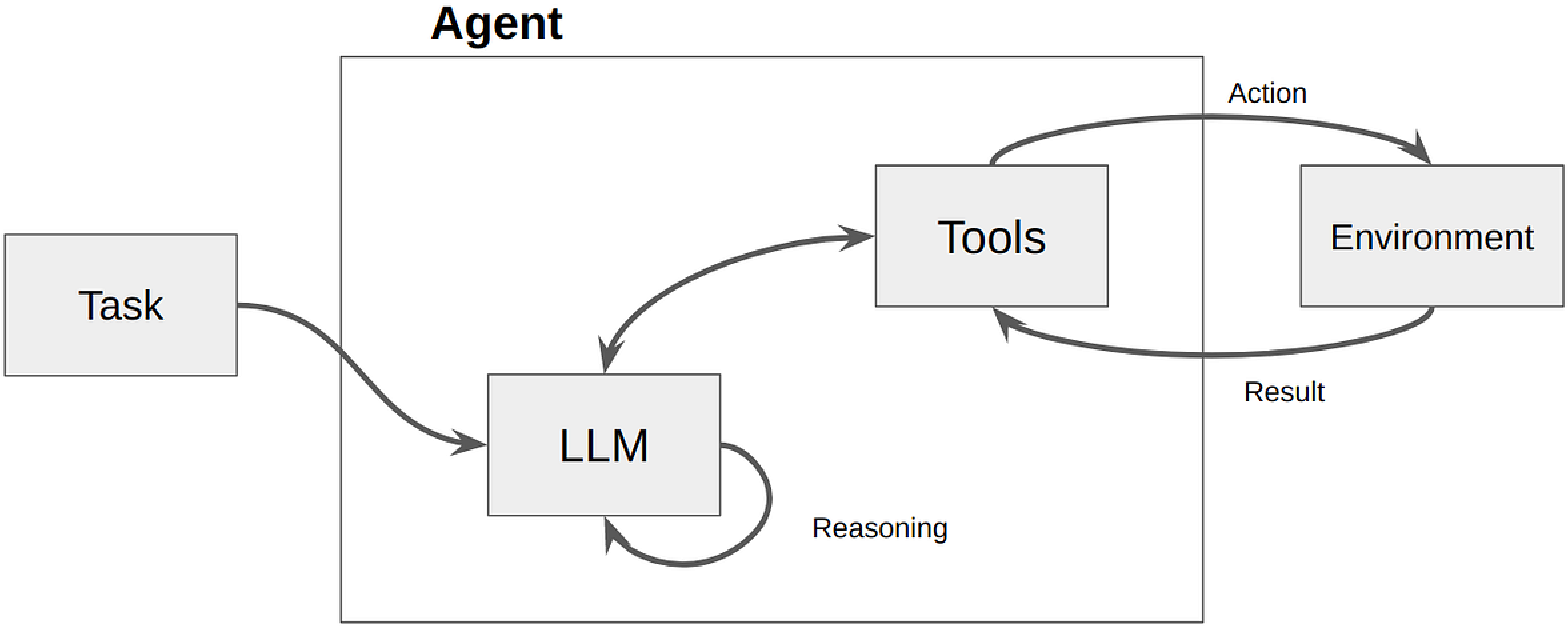
Some applications require an unknown chain of calls to LLMs/other tools. In these types of chains, there is a "agent" which has access to a suite of tools. Depending on the user input, the agent can then decide which, if any, of these tools to call. Chains are hardcoded, an agent reasons on what chain to execute.

Agent Components

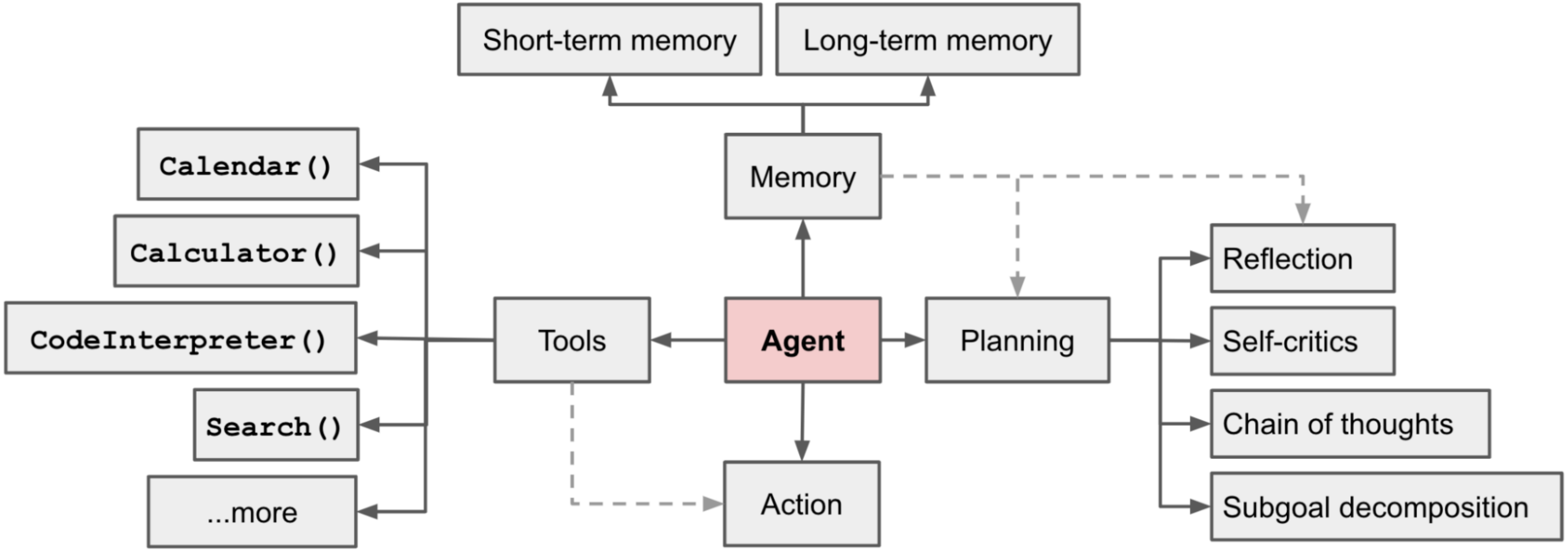
Components of an "agent":

- **Agent** – Wraps a model, takes input, and produces an "action" to take.
- **Tools** – Functions that an agent executes, along with a description of when to use.
- **Memory** – The prior interactions with the agent, such as a "chat history".
- **Agent executor** – combines the agent and tools to perform problem solving.

Agent Components



Agent Components



Use Cases for Chaining and Agents with LangChain

Personal Assistants

1. Prompt Template: Defines the personality of your personal assistant
2. Memory: Equip with short-term conversation retention
3. Tools: Differentiate your assistant by selecting specific capabilities
4. Agent: Design an efficient agent that understands and performs actions.
5. Agent Executor: Establish an environment where the agent can effectively utilize its tools.

Question Answering Over Docs

1. Transform data to a compatible format: Index
2. Ingestion process into a Vectorstore:
3. Load documents using a Document Loader
4. Split documents utilizing a Text Generation
5. Find pertinent documents in the index based on the query
6. Retrieve and return the generated result to the user

Querying Tabular Data

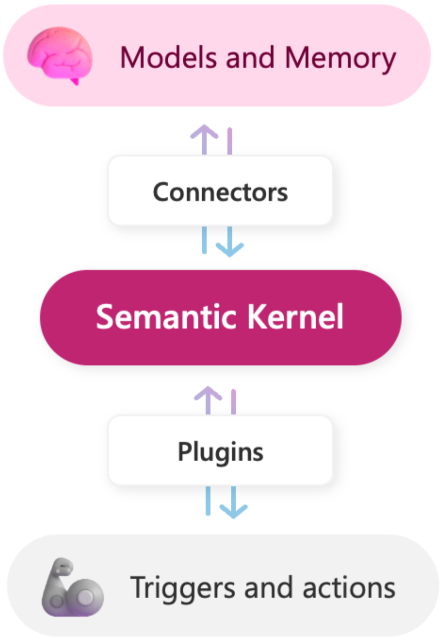
- Tabular data storage: csvs, excel sheets, SQL tables
- Utilize document loaders (e.g., CSVLoader) for loading text data in tabular formats
- Create an index for efficient data querying and interaction
- Direct interaction with numeric tabular data using a language model
- Chain for simple and small datasets
- Agents for complex multi-query processes with the Language Model

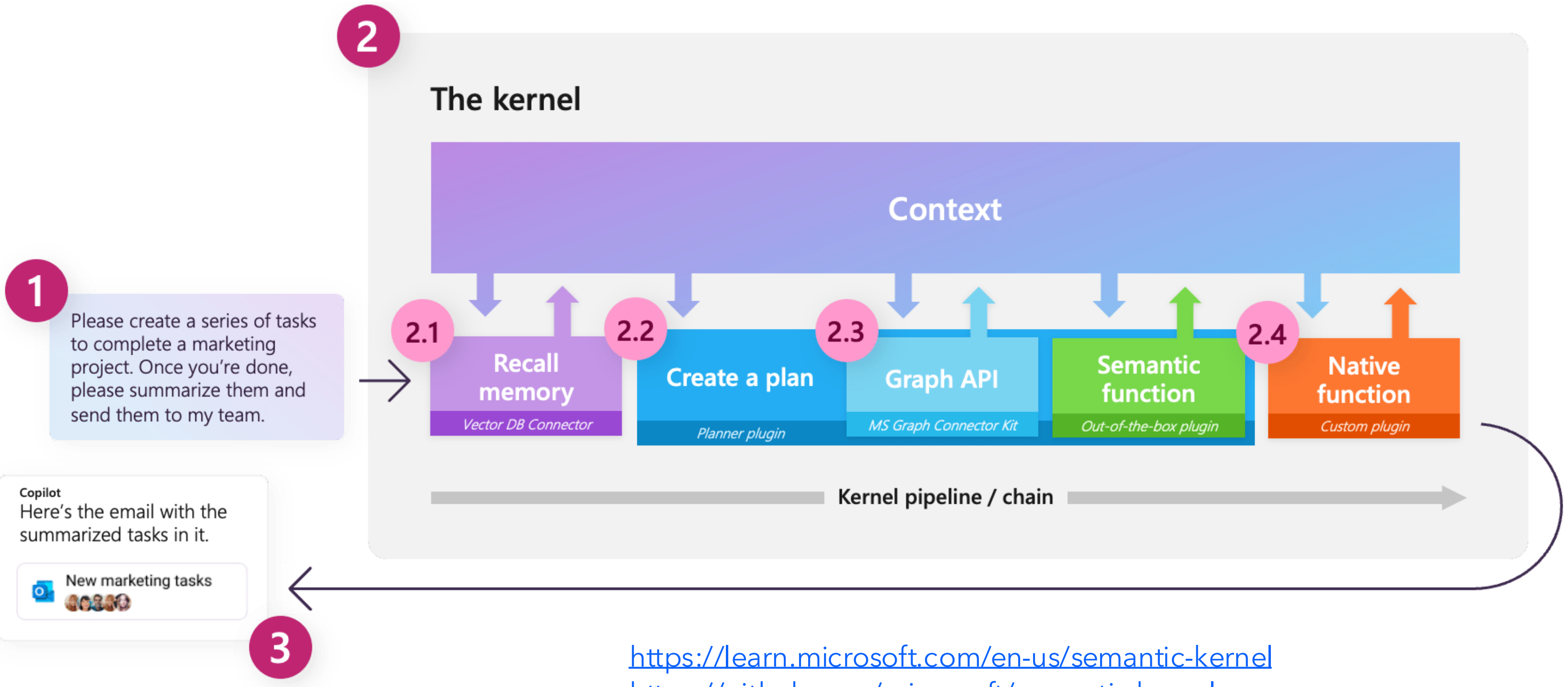
Interacting with APIs

- Integrate LLM with external APIs
- Retrieve context for LLM usage
- Interact with APIs through natural language
- Interface LLMs with external APIs
- Functions (e.g., OpenAI functions)
- LLM-generated interface utilizing API documentation

Semantic Kernel (SK)

- [*Oversimplified*] Microsoft's open-source competitor to LangChain
- Integrate GenAI models, memory, agents, and tools (terminology differs)
- Supported Languages: C#, Python, Java
- "With SK, you can leverage the same AI orchestration patterns that power Microsoft 365 Copilot and Bing in your own apps, while still leveraging your existing development skills and investments."





https://learn.microsoft.com/en-us/semantic-kernel
https://github.com/microsoft/semantic-kernel
https://www.linkedin.com/learning/introducing-semantic-kernel-building-ai-based-apps

03 Chaining in Action

Chaining in Action

Conversation Bot

https://huggingface.co/spaces/kimadams/ai-kit - Prompt Builder Tab

Audio-to-Text:

- Separate audio from video, transcribe audio using Whisper-1 speech recognition

Embedding Query:

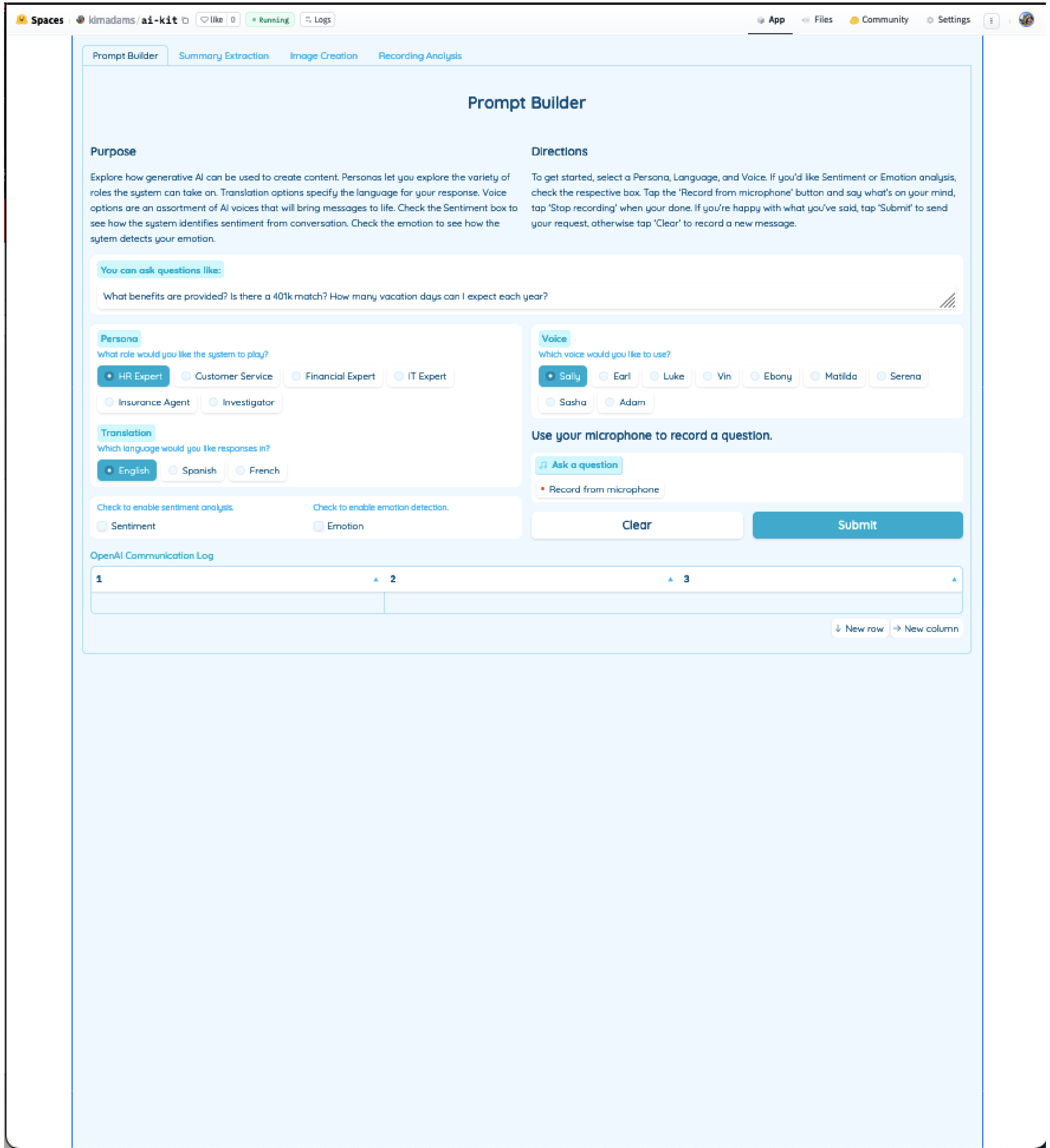
- Leverage embeddings for best answer primer (context)

Assistant Service:

- OpenAI ChatCompletion with prompt and context to push conversation along

AI Voice Service:

- Pass OpenAI response to 11Labs for voice response



Chaining in Action

Recording Analysis

https://huggingface.co/spaces/kimadams/ai-kit - Recording Analysis Tab

Audio-to-Text:

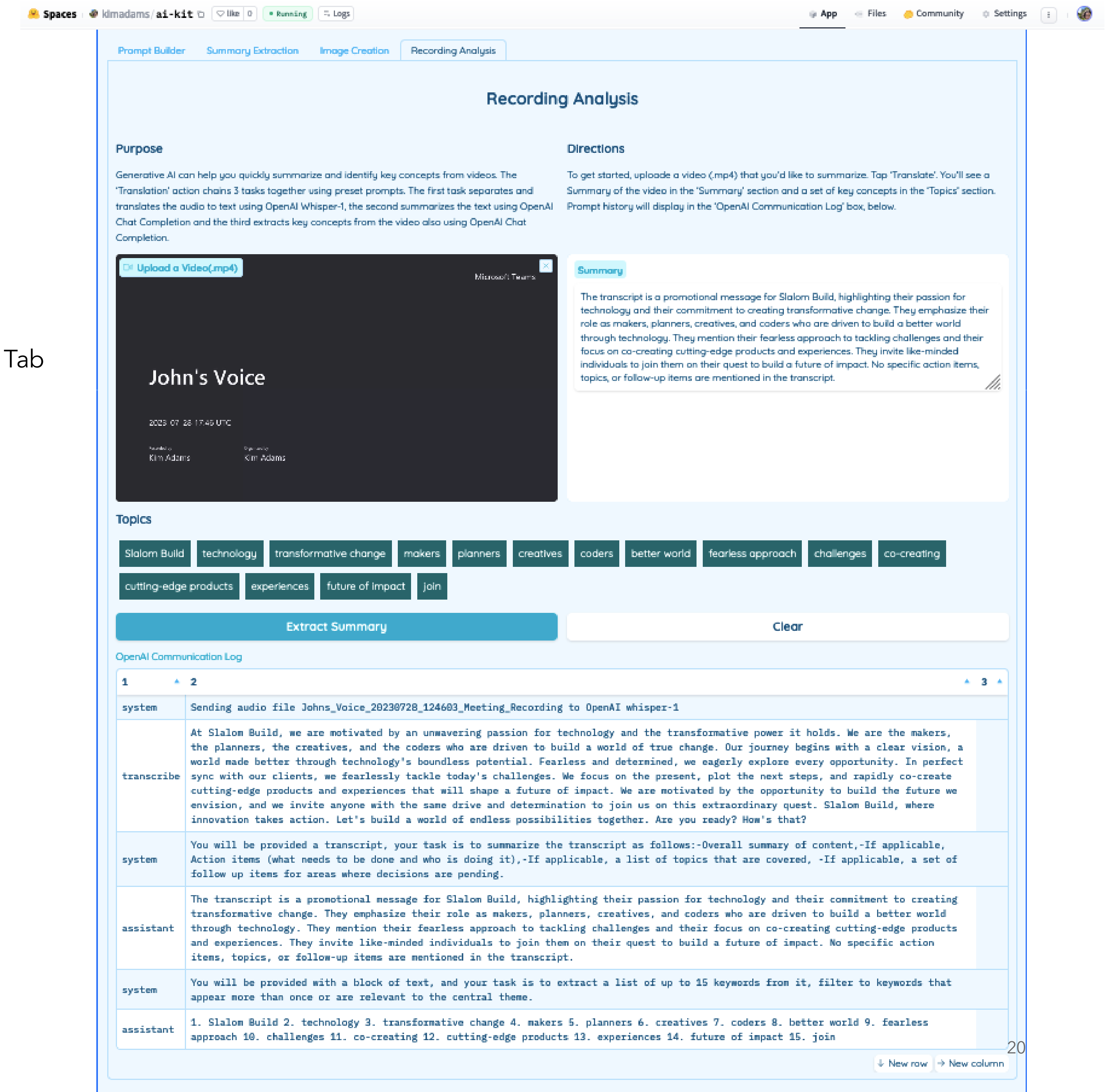
- Separate audio from videos, transcribe audio using Whisper-1 speech recognition

Summarization:

- Condense the transcribed text into clear and concise summaries

Topic Generation:

- Leverage topic modeling algorithms to extract meaningful topics from the summaries



Disambiguation: Chains, Tools, Agents

Chains

- Provide the ability to connect multiple LLM calls and additional steps in a deterministic workflow
- May or may not use tools or agents

Tools

- Enhance LLM capabilities by providing the LLM with additional actions they can take to pull context
- Aka Function Calling

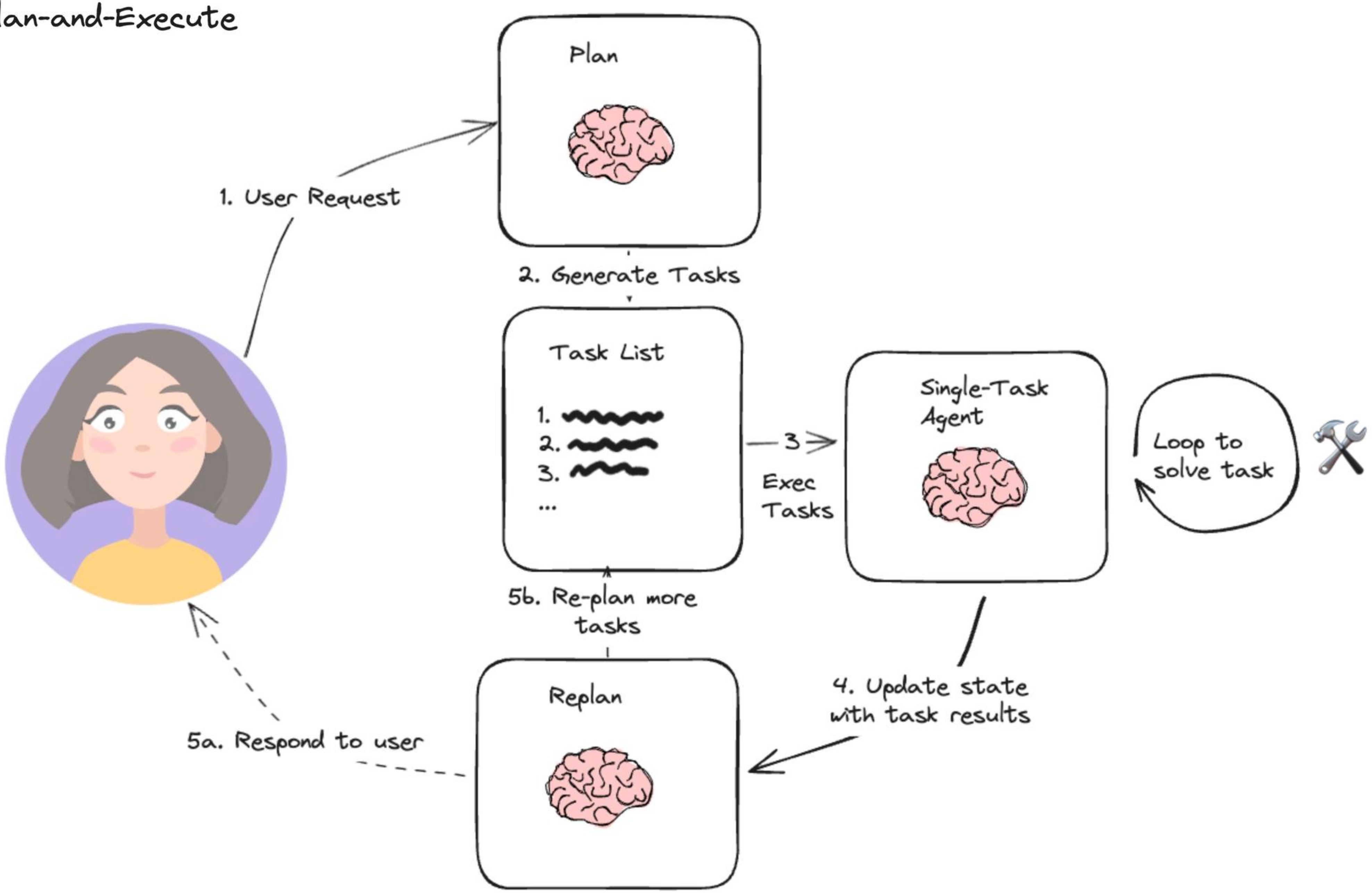
Agents

- Smarter tools that use LLMs to enhance results
- provide a natural language interface for a tool
- Autonomous Agents are smarter Agents that can self-evaluate and run independently from user input or create their own adhoc / non-deterministic workflows

04 Chains and Agents Example

Plan-and-Execute

[Langgraph reference]



Chain Example

```python
def ask_sql_chain(question: str, verbose: bool = True):
    """
    This function runs a SQL query using a chain of operations against a local sample database.

    The database is a sample SQLite database called Chinook. The database contains information about
    a record store. The database has 11 tables: albums, employees, invoices, playlists, artists, genres, media_types,
    tracks, customers, invoice_items, playlist_track

    Parameters:
    question (str): The SQL query to be executed.
    verbose (bool): If True, the function will print additional debug information. Default is True.

    Returns:
    str: The result of the SQL query.
    """

    # Create an instance of SQLDatabase with the database URI and the tables to include.
    db = SQLDatabase.from_uri(
        database_uri=database_uri,
        include_tables=["Album", "Artist", "Track"],
        sample_rows_in_table_info=5,
    )

    llm = AzureChatOpenAI(temperature=0, verbose=True, deployment_name=deployment_name)

    # Create an instance of SQLDatabaseChain with the AzureChatOpenAI instance, the SQLDatabase instance,
    # and the verbose mode set.
    db_chain = SQLDatabaseChain.from_llm(
        llm,
        db,
        verbose=verbose,
    )

    # Run the SQL query and get the result.
    response = db_chain.run(question)

    # Return the result of the SQL query.
    return response


if __name__ == "__main__":
    q = "What is the name of the album that has the most tracks?"
    result = ask_sql_chain(q)
    print(result)
```

```
> Entering new SQLDatabaseChain chain...
What is the name of the album that has the most tracks?
SQLQuery:SELECT "Album"."Title", COUNT("Track"."TrackId") AS "NumTracks"
FROM "Album"
JOIN "Track" ON "Album"."AlbumId" = "Track"."AlbumId"
GROUP BY "Album"."AlbumId"
ORDER BY "NumTracks" DESC
LIMIT 1;
SQLResult: [('Greatest Hits', 57)]
Answer:Greatest Hits
> Finished chain.
Greatest Hits
```

https://bitbucket.org/slalom-consulting/chain-agent-example/src/main/examples/sql_chain.py

Agent Example

```python
def ask_sql_agent(question: str, verbose: bool = False):
    """
    This function creates a sql agent. It can generate and execute one or many SQL queries to
    acquire the data needed to answer the given question against a local sample database.

    The database is a sample SQLite database called Chinook. The database contains information about
    a record store. The database has 11 tables: albums, employees, invoices, playlists, artists, genres, media_types,
    tracks, customers, invoice_items, playlist_track

    Parameters:
    question (str): Question asked in about the database
    verbose (bool): If True, the function will print additional debug information. Default is False.

    Returns:
    str: The result of the SQL query.
    """

    # Create an instance of SQLDatabase with the database URI and the tables to include.
    db = SQLDatabase.from_uri(
        database_uri=database_uri,
        include_tables=["Album", "Artist", "Track", "Customer"],
        sample_rows_in_table_info=5,
    )

    # Create an instance of AzureChatOpenAI with temperature set to 0, verbose mode enabled,
    # and the deployment name set.
    llm = AzureChatOpenAI(
        temperature=0, verbose=verbose, deployment_name=deployment_name
    )

    # Create a SQL agent with the AzureChatOpenAI instance, the SQLDatabase instance, and the verbose mode set.
    agent_executor = create_sql_agent(
        llm=llm,
        toolkit=SQLDatabaseToolkit(db=db, llm=llm),
        verbose=verbose,
        agent_type=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    )

    # Run the SQL query and get the result.
    agent_response = agent_executor.run(question)

    # Return the result of the SQL query.
    return agent_response


if __name__ == "__main__":
    q = "What are the names of the 5 artists with the most albums?"
    result = ask_sql_agent(q)
    print(result)
```

```
> Entering new AgentExecutor chain...
Action: sql_db_list_tables
Action Input: ""
Observation: Album, Artist, Customer, Track
Thought:I can query the Album and Artist tables to find the names of the artists with the most albums. I should check the schema of these tables to see which columns I need to select.
Action: sql_db_schema
Action Input: Album, Artist
Observation:
CREATE TABLE "Album" (
    "AlbumId" INTEGER NOT NULL,
    "Title" NVARCHAR(160) NOT NULL,
    "ArtistId" INTEGER NOT NULL,
    PRIMARY KEY ("AlbumId"),
    FOREIGN KEY("ArtistId") REFERENCES "Artist" ("ArtistId")
)

/*
5 rows from Album table:
AlbumId Title   ArtistId
1   For Those About To Rock We Salute You   1
2   Balls to the Wall   2
3   Restless and Wild   2
4   Let There Be Rock   1
5   Big Ones    3
*/


CREATE TABLE "Artist" (
    "ArtistId" INTEGER NOT NULL,
    "Name" NVARCHAR(120),
    PRIMARY KEY ("ArtistId")
)

/*
5 rows from Artist table:
ArtistId    Name
1   AC/DC
2   Accept
3   Aerosmith
4   Alanis Morissette
5   Alice In Chains
*/
Thought:I can use the ArtistId column from the Album table to join with the Artist table and get the names of the artists. I should order the results by the count of albums in descending order and limit the results to 5.
Action: sql_db_query
Action Input: "SELECT Artist.Name FROM Album JOIN Artist ON Album.ArtistId = Artist.ArtistId GROUP BY Artist.Name ORDER BY COUNT(Album.AlbumId) DESC LIMIT 5"
Observation: [('Iron Maiden',), ('Led Zeppelin',), ('Deep Purple',), ('U2',), ('Metallica',)]
Thought:I now know the final answer.
Final Answer: The names of the 5 artists with the most albums are Iron Maiden, Led Zeppelin, Deep Purple, U2, and Metallica.

> Finished chain.
The names of the 5 artists with the most albums are Iron Maiden, Led Zeppelin, Deep Purple, U2, and Metallica.
```

https://bitbucket.org/slalom-consulting/chain-agent-example/src/main/examples/sql_agent.py

05 Challenge Project

Challenge Project

<u>Demo at Week 7 session</u>

- If you can't make it, record your demo!
- Get creative
- Demonstrate one or more skills that you've learned
- Doesn't need to be polished or have a UI
  - Python notebooks are fine
  - Command line is fine
- Individuals or small teams
- Sign up with your team (or solo) → [Demo Sign-up Sheet]
  - (so that we can plan enough time for demos)

<u>Sample Ideas:</u>

- Soccer rules chat
- Generate a kitten picture based on the weather

Thank You

Resources

YouTube (Sam Witteveen):

- LangChain – Basics: https://www.youtube.com/watch?v=J_0qvRt4LNk
- LangChain – Tools & Chains: https://www.youtube.com/watch?v=hI2BY7yl_Ac
- LangChain – Conversations with Memory: https://www.youtube.com/watch?v=X550Zbz_ROE
- LangChain – Agents: https://www.youtube.com/watch?app=desktop&v=ziu87EXZVUE

LangChain Memory

Assists in providing "memory" to LLM interactions

ConversationBufferMemory

Buffers the entire conversation which can be retrieved programmatically. Best for limited number of interactions given token restrictions.

ConversationBufferWindowMemory

Similar to ConversationBufferMemory but limits number of previous interactions. May run into token issues if configured for high number of interactions.

ConversationSummaryMemory

Instead of taking the last X interactions, continues to summarize the interactions so far. Takes more tokens initially, but less as the conversation extends.

ConversationKGMemory

Builds a Knowledge Graph as the conversation is happening.

Entity Memory

Extracts information keyed by entities

Custom Memory

Combines different types of memories

Why Memory?

- *Passing the previous context for a long-running conversation may run into token issues*
- *People expect Bots to remember long conversations*
- *Co-reference resolution during natural conversation. Example: "My friend Andrew…" later referencing Andrew by "him"*