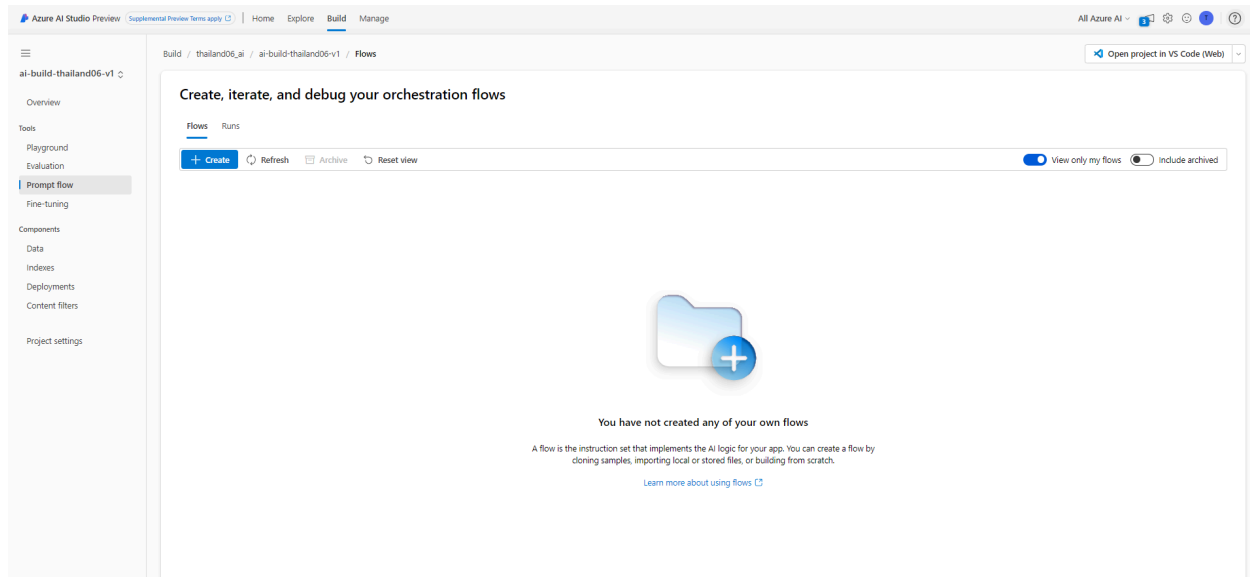
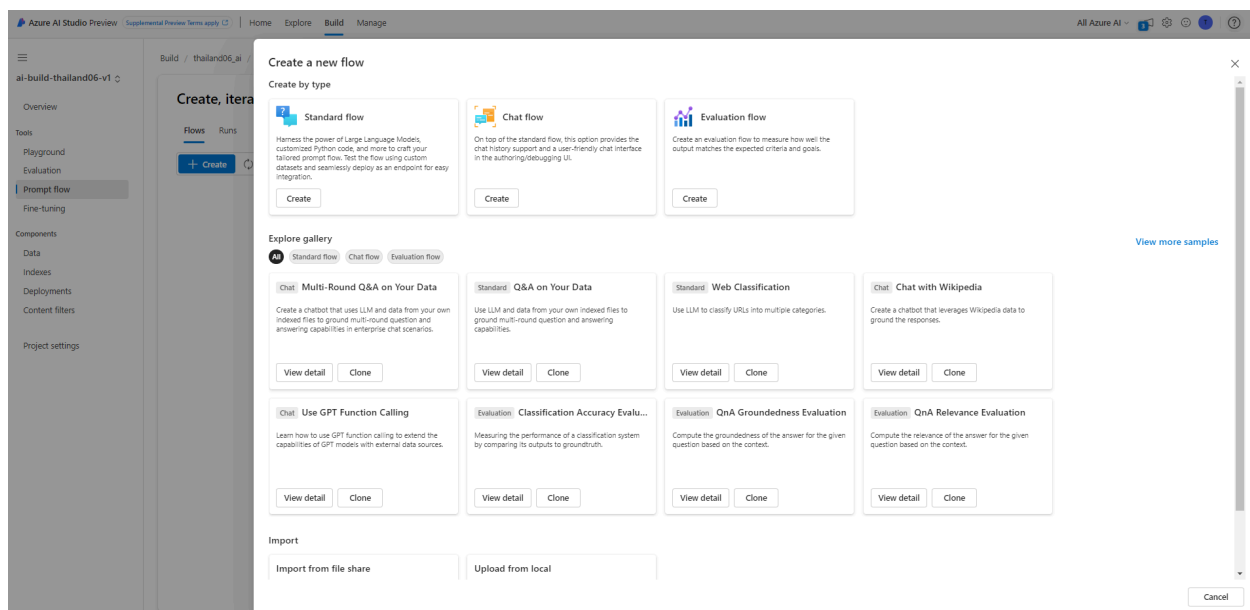


Prompt flow (Multi-Turn RAG)

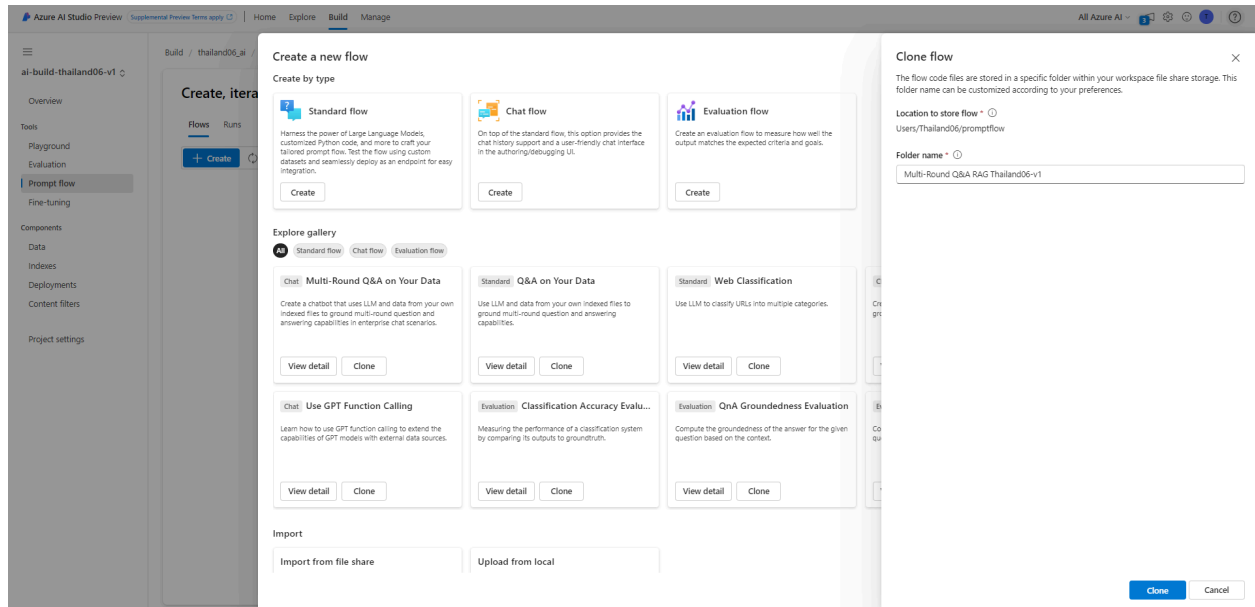
1. Go to prompt flow and click **“create”** to create a new prompt flow. You will find specific types of flows to be made and some of the **“Example Flows,”** which you can clone and explore for further understanding.



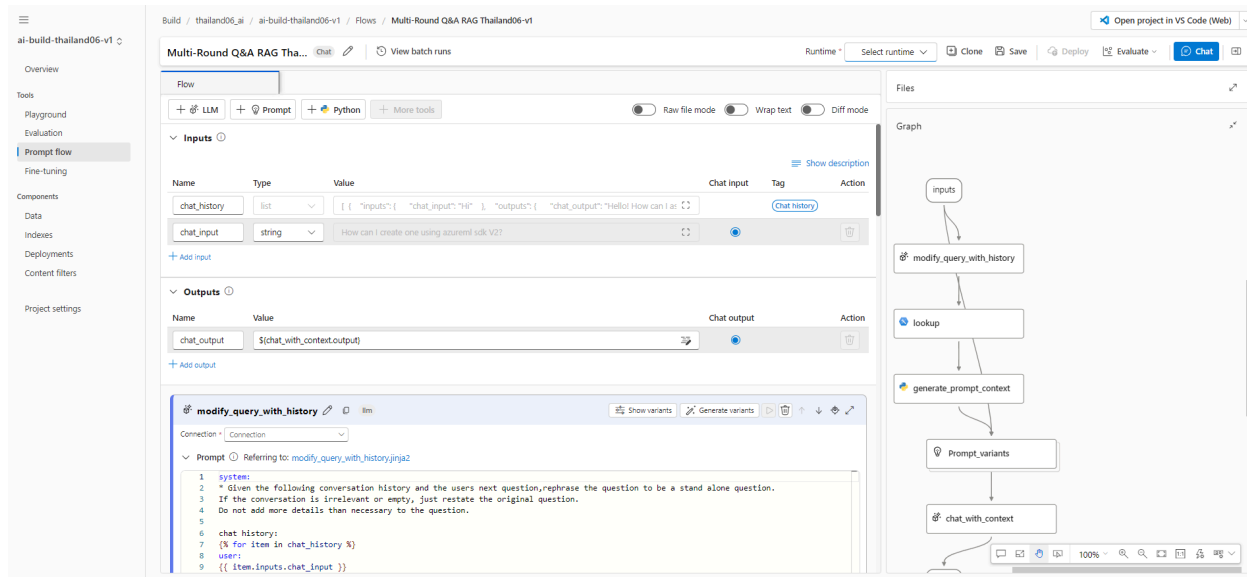
2. We will select **“Multi-Round Q&A on Your Data”** for this example. Click on **“Clone”**



- Specify a folder name for your project. You can use the following naming convention **“Multi Round Q&A RAG” + login ID + version number**. E.g. **“Multi Round Q&A RAG Thailand06-v1”**

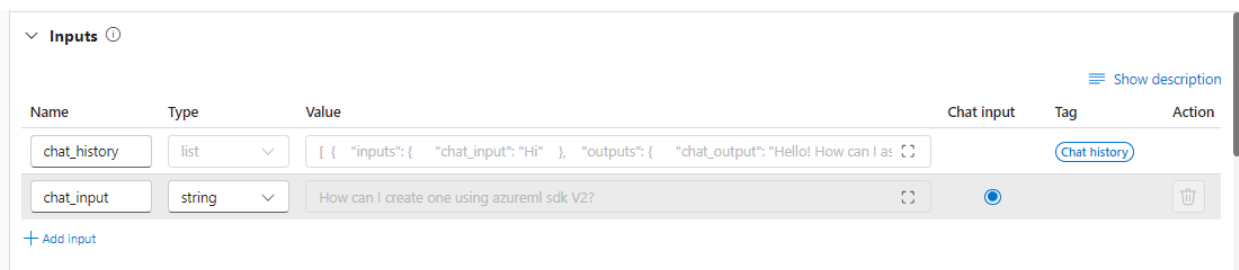


- In the prompt flow, you can see the **“graph,”** which gives you a bird's-eye view of the whole flow, and the **“Files,”** which lets you download the prompt flow files to share your work. Click on the dropdown option next to **Runtime** & select **Start**.



We can now start setting up the components under the **Flow** tab. Note: You can add/remove components from the flow as per the requirements.

- Under the **Inputs** section, you can add more than one input and specify different data types for each type. In this example, **chat_input** is an input name with **string** as the data type.



6. Under the **Outputs** section, you can specify the number of outputs and even assign values from other components of the flow. In this example, **chat_output** is the output name and **`\${chat_with_context.output}`** is the value. Here **chat_with_context** is another component of the flow and **output** is a variable of that component.

Name	Value	Chat output	Action
chat_output	<code>\${chat_with_context.output}</code>	<input checked="" type="checkbox"/>	

+ Add output

7. Under **modify_query_with_history** we would already have the prompts to create standalone questions from conversation history. We can tweak the prompts as needed.

modify_query_with_history

Connection: Connection

▼ Prompt Referring to: [modify_query_with_history.jinja2](#)

```
1 system:
2 * Given the following conversation history and the users next question, rephrase the question to be a stand alone question.
3 If the conversation is irrelevant or empty, just restate the original question.
4 Do not add more details than necessary to the question.
5
6 chat history:
7 {% for item in chat_history %}
8 user:
9 {{ item.inputs.chat_input }}
10 assistant:
11 {{ item.outputs.output }}
12 {% endfor %}
13
14 Follow up Input: {{ chat_input }}
15 Standalone Question:
```

▼ Inputs

Name	Type	Value
chat_history	string	<code>\${inputs.chat_history}</code>
chat_input	string	<code>\${inputs.chat_input}</code>

> Activate config

Set up the OpenAI connection in **modify_query_with_history**. Click on the drop-down next to **Connection** and select **openai-tigeranalytics-<number>**. In **deployment_name** select **gpt-35-turbo-16k**. In **response_format** select **`\${"type": "text"}`**.

modify_query_with_history llm Show variants Generate variants Play Trash Up Down Refresh Link

Connection: openai-tigeranalytics-101 Api: chat

deployment_name: gpt-35-turbo-16k temperature: 0 stop: max_tokens: 100 response_format: ["type":"text"]

> Advanced

> Function calling

✓ Prompt ⓘ Referring to: [modify_query_with_history.jinja2](#)

```

1 system:
2 * Given the following conversation history and the users next question, rephrase the question to be a stand alone question.
3 If the conversation is irrelevant or empty, just restate the original question.
4 Do not add more details than necessary to the question.
5
6 chat history:
7 {% for item in chat_history %}
8 user:
9 {{ item.inputs.chat_input }}
10 assistant:
11 {{ item.outputs.output }}
12 {% endfor %}
13
14 Follow up Input: {{ chat_input }}
15 Standalone Question:

```

✓ Inputs Validate and parse input

Name	Type	Value
chat_history	string	\$(inputs.chat_history) Copy
chat_input	string	\$(inputs.chat_input) Copy

8. The **lookup** tab is for the retrieval system which uses Azure AI search service. Set up the connection to the created index by clicking on the **Value** box in the **mlindex_content**.

lookup Index Lookup Preview Play Trash Up Down Refresh Link

✓ Inputs

Name	Type	Value
mlindex_content	string	 Copy
queries	object	\$(modify_query_with_history.output) Copy
query_type	string	 Copy
top_k	int	2 Copy

> Activate config

Under **index_type**, select **Registered Index**, the index already created in the playground setup.

Generate

Name	Type	Value
index_type	string	<div> <div></div> <div>Refresh</div> <div>Registered Index</div> <div>Azure AI Search</div> <div>FAISS</div> <div>MLIndex file from path</div> </div>

Select the index name under **mlindex_asset_id** that is already created. The create index name would be **“ai-build-” + user_id + version**. E.g. **ai-build-thailad06-index-v1**

Generate

Name	Type	Value
index_type	string	Registered Index
mlindex_asset_id	string	ai-build-thailand06-index-v1:1

Select **Hybrid (vector + keyword)** under **query_type**. You can select other search types as well based on the use case. Specify **top_k** for the number of chunks you want to fetch as part of the search.

lookup

Index Lookup

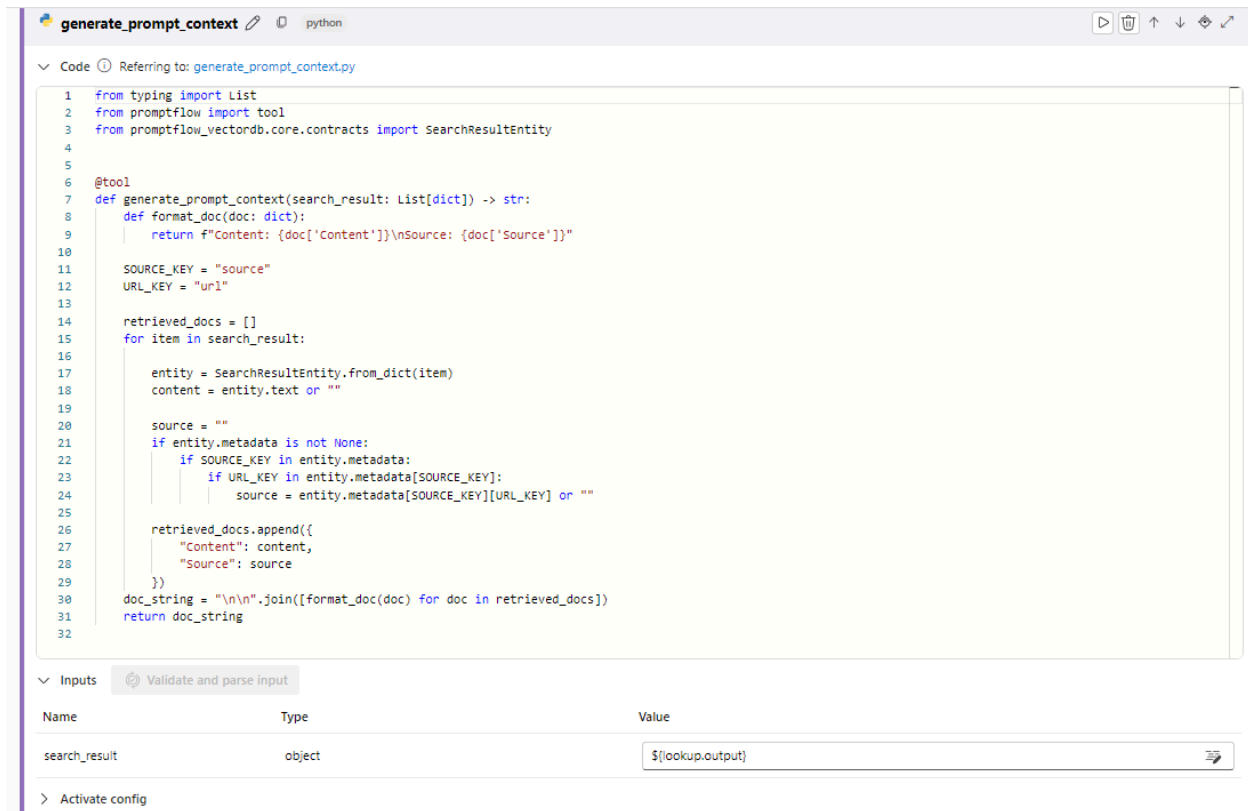
Preview

Inputs

Name	Type	Value
mlindex_content	string	embeddings: api_base: https://openai-tigeranalytics-101.openai.azure.com/
queries	object	\$(modify_query_with_history.output)
query_type	string	Hybrid (vector + keyword)
top_k	int	2

Activate config

9. The **generate_prompt_context** tab is Python code that combines the context and source retrieved by Azure AI search.



The screenshot shows a code editor window titled "generate_prompt_context" with a Python file icon. The code is as follows:

```
1 from typing import List
2 from promptflow import tool
3 from promptflow_vectordb.core.contracts import SearchResultEntity
4
5
6 @tool
7 def generate_prompt_context(search_result: List[dict]) -> str:
8     def format_doc(doc: dict):
9         return f"Content: {doc['Content']}\nSource: {doc['Source']}"
10
11     SOURCE_KEY = "source"
12     URL_KEY = "url"
13
14     retrieved_docs = []
15     for item in search_result:
16
17         entity = SearchResultEntity.from_dict(item)
18         content = entity.text or ""
19
20         source = ""
21         if entity.metadata is not None:
22             if SOURCE_KEY in entity.metadata:
23                 if URL_KEY in entity.metadata[SOURCE_KEY]:
24                     source = entity.metadata[SOURCE_KEY][URL_KEY] or ""
25
26         retrieved_docs.append({
27             "Content": content,
28             "Source": source
29         })
30     doc_string = "\n\n".join([format_doc(doc) for doc in retrieved_docs])
31     return doc_string
32
```

Below the code editor, there is an "Inputs" section with a "Validate and parse input" button. It contains a table with the following data:

Name	Type	Value
search_result	object	<code>\${lookup.output}</code>

At the bottom of the Inputs section, there is a link to "Activate config".

10. The **Prompt_variants** section combines the chat history, context, and system prompts to create a common prompt template to be fed to LLM for generating responses. We can tweak the template and perform prompt engineering as needed.

Prompt_variants

prompt

3 variants Current: variant_0

Show variants

Generate variants

▼

Prompt

Referring to: Prompt_variants.jinja2

```
1 system:
2 * You are an AI system designed to answer questions from users in a designated context. When presented with a scenario, you must reply with accuracy to inquirers
3 Please add citation after each sentence when possible in a form "(Source: citation)".
4 context: {{contexts}}
5
6 chat history:
7 {% for item in chat_history %}
8 user:
9 {{ item.inputs.chat_input }}
10 assistant:
11 {{ item.outputs.chat_output }}
12
13 {% endfor %}
14
15 user:
16 {{ chat_input }}
```

▼

Inputs

Validate and parse input

Name	Type	Value
contexts	string	<div>\$(generate_prompt_context.output)</div>
chat_history	string	<div>\$(inputs.chat_history)</div>
chat_input	string	<div>\$(inputs.chat_input)</div>

> Activate config

11. **Chat_with_context** takes the prompt from the above step and generates responses using LLM.

chat_with_context

llm

Show variants

Generate variants

Connection *

Connection

▼

▼

Prompt

Referring to: chat_with_context.jinja2

```
1 {{prompt_text}}
2
3
```

▼

Inputs

Validate and parse input

Name	Type	Value
prompt_text	string	<div>\$(Prompt_variants.output)</div>

> Activate config

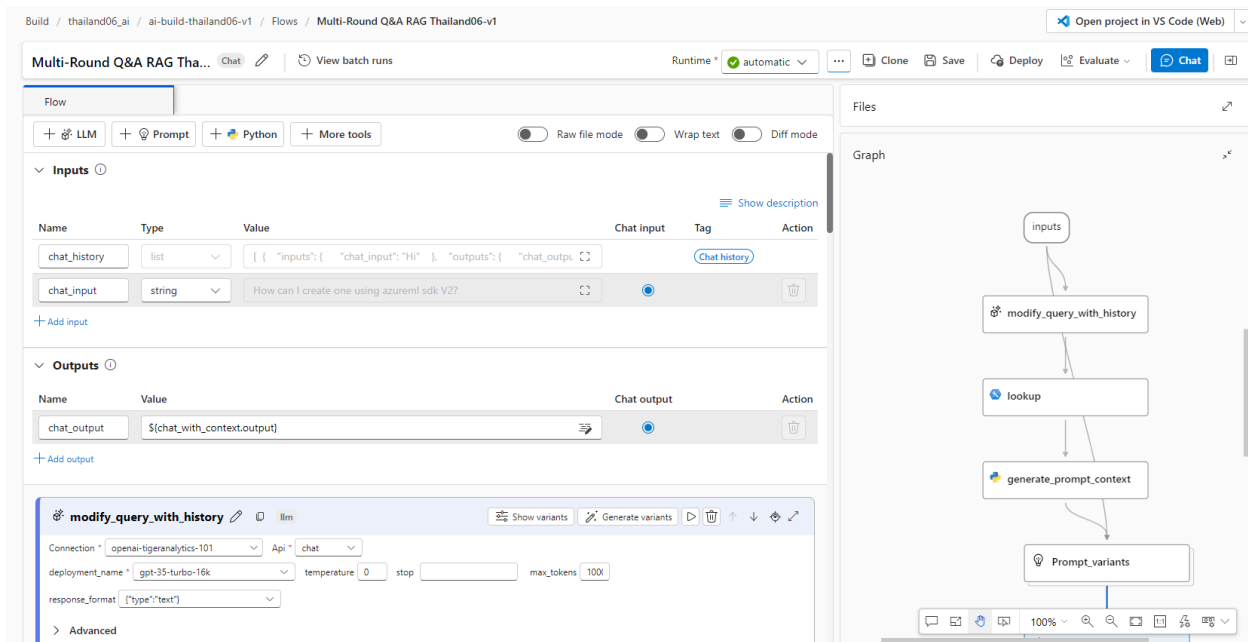
Set up the OpenAI connection in **chat_with_context**. Click on the drop-down next to **Connection** and select **openai-tigeranalytics-<number>**. In **deployment_name** select **gpt-35-turbo-16k**. In **response_format** select **{"type": "text"}**.

The screenshot shows the configuration interface for a chat model. At the top, there are tabs for 'Show variants', 'Generate variants', and a play button. Below this, the 'Connection' is set to 'openai-tigeranalytics-101' and the 'Api' is 'chat'. The 'deployment_name' is 'gpt-35-turbo-16k', 'temperature' is '0', 'stop' is empty, 'max_tokens' is '100', and 'response_format' is '({type:"text"})'. There are expandable sections for 'Advanced', 'Function calling', and 'Prompt'. The 'Prompt' section is expanded, showing a text area with a placeholder '1 {{prompt_text}}'. Below the prompt, there is an 'Inputs' section with a table:

Name	Type	Value
prompt_text	string	\$(Prompt_variants.output)

At the bottom, there is an 'Activate config' button.

12. The prompt flow is ready to use and you can view the process flow from the **Graph** tab on the right section.



You can click the **Chat** option to open the chat window and interact with the application.

The screenshot displays the 'Multi-Round Q&A RAG Thailand06-v1' application interface. The top navigation bar includes 'Build / thailand06_ai / ai-build-thailand06-v1 / Flows / Multi-Round Q&A RAG Thailand06-v1' and a button to 'Open project in VS Code (Web)'. The main interface is divided into two sections: 'Flow' and 'Chat'.

Flow Configuration:

- Inputs:** A table with columns 'Name', 'Type', 'Value', 'Chat input', 'Tag', and 'Action'. It lists 'chat_history' (list) and 'chat_input' (string). The 'chat_input' value is 'What is the percentage of energy saved in recycling aluminum cans <'. There is a '+ Add input' button.
- Outputs:** A table with columns 'Name', 'Value', 'Chat output', and 'Action'. It lists 'chat_output' with the value '\$[chat_with_context.output]'. There is a '+ Add output' button.
- LLM Configuration:** A section titled 'modify_query_with_history' with fields for 'Connection' (openai-tigeranalytics-101), 'Api' (chat), 'deployment_name' (gpt-35-turbo-16k), 'temperature' (0), 'stop' (empty), 'max_tokens' (100), and 'response_format' (['type':'text']). There is an 'Advanced' link below.

Chat Window:

- Question 1:** 'How many trees can be saved by recycling one ton of paper?'.
Answer: 'According to the information provided in the context, recycling one ton of paper can save approximately 17 trees. (Source: Recycling Guide Chapter 2.pdf)'.
Token Info: 'Total tokens for generating this: 1526 tokens, time spent: 9.52 sec'.
- Question 2:** 'What is the percentage of energy saved in recycling aluminum cans compared to producing them from virgin materials?'.
Answer: 'According to the information provided in the context, recycling aluminum cans can save 95% of the energy required to produce the same amount of aluminum from virgin materials. (Source: Recycling Guide Chapter 1.pdf)'.
Token Info: 'Total tokens for generating this: 1626 tokens, time spent: 9.88 sec'.
- Input field:** 'Input anything to test...'.