Microsoft

Azure OpenAI Fed Civ Workshop

Cameron Jackson

Azure OpenAI Playground - Understanding the Models Available

- In this section you will gain:
  - Awareness of the Large Language Models available in the Azure OpenAI Playground.
  - In-depth information on the models that are most efficient.

Overview of Large Language Models

Azure OpenAI is home to the most diverse set of large language models with different capabilities and price points. Each model has been pre-trained on complex datasets as of the year 2021 and can be easily integrated into the Playground. Alongside integrating into the Playground, you can also integrate these models into your custom applications. The models available include:

- **GPT -4**
  - A set of models that improve on GPT-3.5 and can understand as well as generate natural language and code.
- **GPT -3.5**
  - A set of models that improve on GPT-3 and can understand as well as generate natural language and code.
- **Embeddings**
  - A set of models that can convert text into numerical vector form to facilitate text similarity.

In-Depth of GPT-4

GPT-4

- **GPT-4**: OpenAI’s most accurate problem-solving model
- **GPT-4**: Chat-optimized and good at traditional completion tasks such as text generation, summarization, translation, etc.
- **GPT-4**: Accessible via Chat Completions API

In-Depth of GPT-3.5

GPT-3.5

- **GPT-3.5 Models**: Natural language and code experts
- **GPT-3.5 Turbo**: The most advance model of the GPT-3.5 series.
- **GPT-3.5 Turbo**: Chat-friendly and good at traditional completion tasks such as text generation, summarization, translation, etc.
- **GPT-3.5 Turbo**: Recommended over GPT-3.5 and GPT-3 models.

In-Depth of Embeddings

Embeddings

- **Embeddings**: Numerical text representations for relatedness measurement
- **Embeddings**: Useful for search, clustering, recommendations, anomaly detection, and classification tasks
- **text-embedding-ada-002**: Second-generation embedding model that replaces 16 first-generation models.
- **text-embedding-ada-002**: Highly efficient compared to its predecessors.

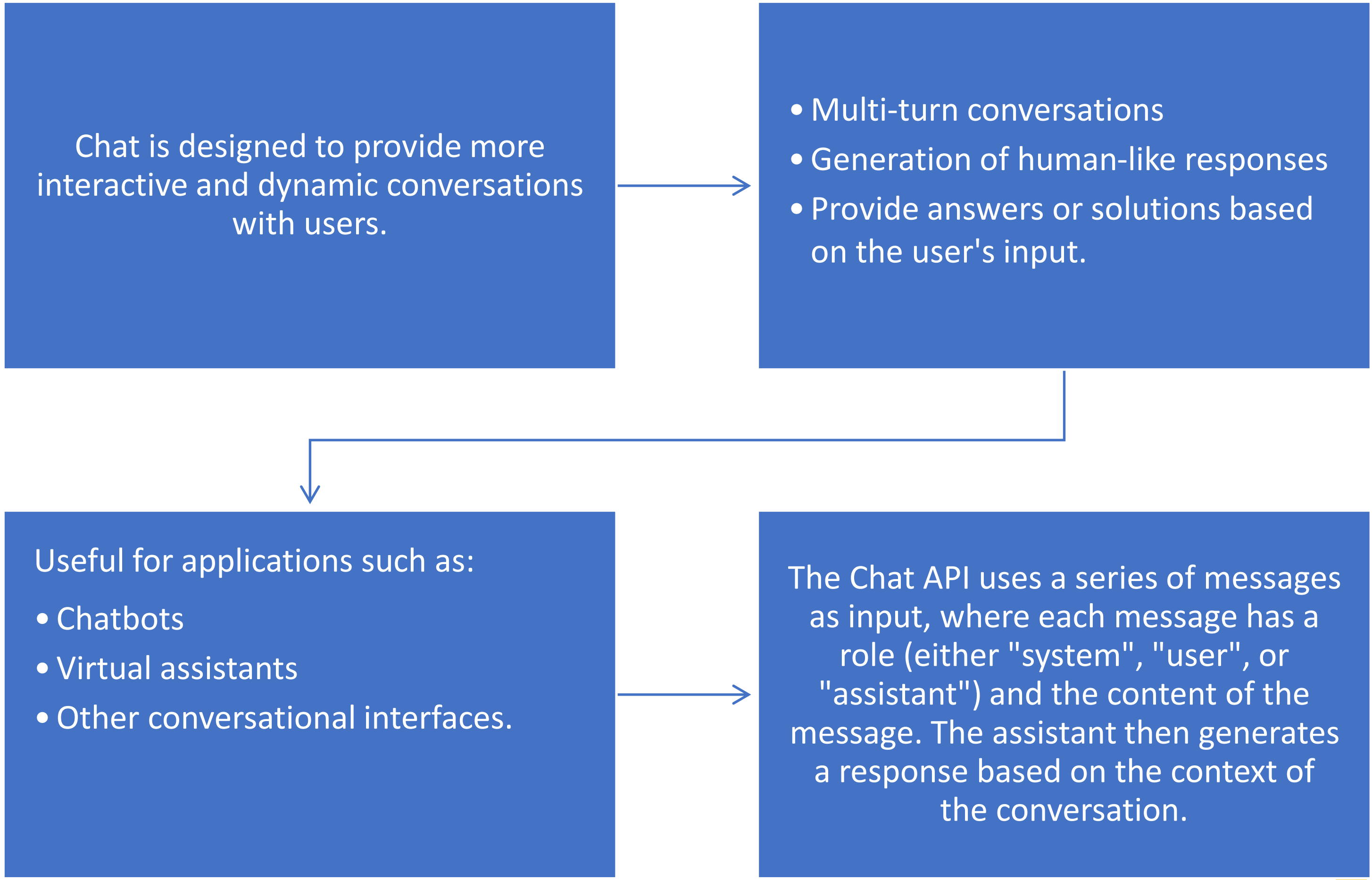
Large Language Models: Pricing, Token Limits, Regions, and Fine Tuning

Visit: [Azure OpenAI Service models - Azure OpenAI | Microsoft Learn]()

Azure OpenAI Playground - Key Components of the Interface

- In this section you will obtain knowledge in:
  - Chat Completions
  - Adding your data
  - Parameters

Overview of Chat API



Chat is designed to provide more interactive and dynamic conversations with users.

- Multi-turn conversations
- Generation of human-like responses
- Provide answers or solutions based on the user's input.

Useful for applications such as:

- Chatbots
- Virtual assistants
- Other conversational interfaces.

The Chat API uses a series of messages as input, where each message has a role (either "system", "user", or "assistant") and the content of the message. The assistant then generates a response based on the context of the conversation.

Adding Data to the Playground

Parameters of Chat Completions

- Max Response
- Temperature
- Top probabilities (Top P)
- Stop Sequence
- Frequency Penalty
- Presence Penalty
- Current Token Count

Parameter: Max Response

- **Max Response:** Specifies the maximum number of tokens (words, characters, or other language units) to be generated in the prompt and response.

Important to choose an appropriate value for max response to ensure that:

- *The response generated is both informative and concise.*
- *Fits within the model's maximum token limit.*

Parameter: Temperature

- **Temperature**: controls the randomness and creativity of the generated text.
- **Includes a range from 0 to 1:**
  - A value closer to 1 result in more diverse and random outputs.
  - A value closer to 0 result in more deterministic and focused outputs.

Optimal temperature value depends on your specific use case and the desired balance between randomness and coherence in the generated text.

Parameter: Top probabilities (Top P)

- **Top P**: controls the randomness of the generated text by sampling from the model's output distribution.
- **Includes a range from 0 to 1:**
  - A value closer to 1 results in more diverse and random outputs.
  - A value closer to 0 results in more focused and deterministic outputs.

Top P is an alternative to temperature for controlling the randomness of the generated text. While both parameters influence the output, they have different mechanisms:

- *Temperature: adjusts the overall distribution of probabilities for all tokens*
- *Top P: sets a threshold for the cumulative probability of the selected tokens.*

Parameter: Stop Sequence

- **Stop sequence**: a specific set of characters or a phrases that instruct the model to stop generating text once it encounters the specified sequence.

Useful for generating single-line answers or short responses.

Parameter: Frequency Penalty

- **Frequency penalty**: controls the output of the model based on the rarity of the tokens generated.
- **Includes a range from 0 to 1:**
  - A value close to 0 makes the model more likely to generate rare words or phrases.
  - A value close to 1 encourages the model to produce more common words or phrases.

Setting the frequency penalty too low may result in:

- *outputs that are difficult to understand*
- *use less common language*

Parameter: Presence Penalty

- **Presence penalty**: controls the likelihood of the model repeating words or phrases in its output.
- **Includes a range from 0 to 1:**
    - A value close to 0 increases the model to repeat words or phrases more often.

    - A value close to 1 reduces the model from using the same words or phrases frequently.

Finding the right balance for presence penalty ensures that the generated content remains informative, engaging, and coherent.

Parameter: Current Token Count

- **Current Token Count**: total number of tokens in each API call, including both input and output tokens.
- ***Tokens*** are used to measure the length of the text being processed or generated.
  - They can be a single character, word, or any other unit of text, depending on the language being used.
- *It is important to be aware of the current token count because there is a maximum token limit for each engine.*
- *Effects the cost of API usage, as you are billed based on the number of tokens processed.*
- *Influences the response time, as generating more tokens may take longer.*

Basics of Prompt Engineering - Introduction to Prompt Engineering

- In this section you will obtain knowledge in:
  - Prompt engineering
  - Importance of well-crafted prompts
  - Common type of prompts

Prompt Engineering

Prompt Engineering – the process of carefully designing LLM prompts to generate a specific output.

Prompts play a crucial role in:

obtaining optimal results from the LLM

Knowing how to write can make a significant difference in the output generated.

Importance of well-crafted prompts

Provide	provide clear and concise instructions or questions that guide the LLM towards providing a relevant and useful answer.
Eliminate	eliminate confusion or ambiguity about what is being asked.
Ensure	ensure that the LLM is generating responses based on the data it is given.
Encourage	encourage the LLM to think more deeply about their answers and provide more thoughtful responses.
Well	overall, well-crafted prompts are essential for ensuring that the models respond with accurate, relevant, and useful information. They also help to improve the quality and usefulness of research and survey data.

Types of Prompts

What are the types of prompts:

- Zero-shot
- Few-shot
- Chain-of-thought

Zero-shot

- Prompting technique that does not require training examples.

- Responses generated based on the prompt it was given.

- Useful in scenarios where there is no specific training data.
  - Can generate results based on general knowledge and assumptions.
- Ex:

  System Message: Classify the text into neutral, negative or positive.
  Query: I think the vacation is okay.

Few-shot

- Prompting technique that involves giving a small number of examples of the desired outputs.
  - With few examples, the model can learn the patterns in the data and generate appropriate results.
- Useful when you want to obtain specific outputs.
- Ex:

System Message: You are an assistant that uses new words in a sentence.

Example: A "whatpu" is a small, furry animal native to Tanzania.

An example of a sentence that uses the word whatpu is: We were traveling in Africa, and we saw these very cute whatpus.

Query: To do a "farduddle" means to jump up and down really fast.

An example of a sentence that uses the word farduddle is:

Chain of Thought

- Prompting technique used to enable complex reasoning capabilities through intermediate reasoning steps.
- Can be combined with few-shot prompting for better results.
- Useful for tasks that require reasoning before responding.
- Ex:

  System Message: Calculate if all the numbers add to an even number.

  Example: The odd numbers in this group add up to an even number: 4, 8, 9, 15, 12, 2, 1.

  A: Adding all the odd numbers (9, 15, 1) gives 25. The answer is False.

  Query: The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.

Basics of Prompt Engineering – Strategies for Effective Prompt Engineering

- In this section you will obtain knowledge in the:
  - Processes of creating prompts
  - Practices for creating prompts

Iterative Design Process and Best Practices for Creating Prompts

1. Write clear instructions

- Include details in your query to get more relevant answers
- In order to get a highly relevant response, make sure that requests provide any important details or context. Otherwise, you are leaving it up to the model to guess what you mean.

Iterative Design Process and Best Practices for Creating Prompts

1. Write clear instructions

- Ask the model to adopt a persona
- The system message can be used to specify the persona used by the model in its replies.

Iterative Design Process and Best Practices for Creating Prompts

1. Write clear instructions

- Use delimiters to clearly indicate distinct parts of the input
- Delimiters like triple quotation marks, XML tags, section titles, etc. can help demarcate sections of text to be treated differently.

Iterative Design Process and Best Practices for Creating Prompts

1. Write clear instructions

- Specify the steps to complete a task
- Some tasks are best specified as a sequence of steps. Writing the steps out explicitly can make it easier for the model to follow them.

Iterative Design Process and Best Practices for Creating Prompts

1. Write clear instructions

- Provide examples
- Providing general instructions that apply to all examples is generally more efficient than demonstrating all permutations of a task by example, but in some cases providing examples may be easier.

Iterative Design Process and Best Practices for Creating Prompts

1. Write clear instructions

- Specify desired length of the output
- You can ask the model to produce outputs that are of a given target length. Can be specified in terms of the count of words, sentences, paragraphs, bullet points, etc.
  - Note: instructing the model to generate a specific number of words does not work with high precision. The model can more reliably generate outputs with a specific number of paragraphs or bullet points.

Iterative Design Process and Best Practices for Creating Prompts

2. Provide reference text

- Instruct the model to answer using a reference text
- If we can provide a model with trusted information that is relevant to the current query, then we can instruct the model to use the provided information to compose its answer.
  - Since GPTs have limited context windows, we need a way to dynamically look up information that is relevant to the question being asked. Embeddings can be used as a solution to implement efficient knowledge retrieval.

Iterative Design Process and Best Practices for Creating Prompts

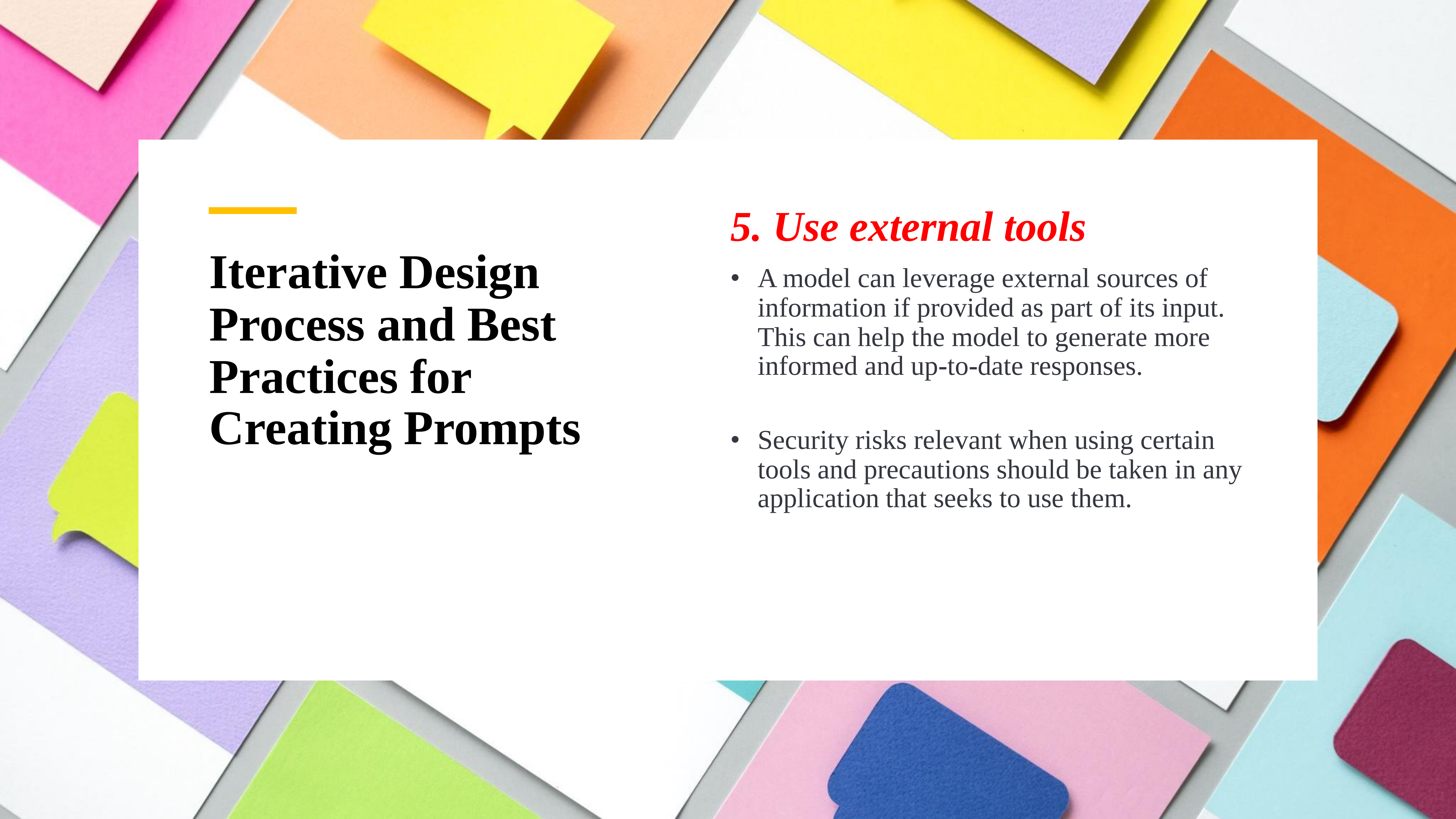
3. Split complex tasks into simpler subtasks

- Use intent classification to identify the most relevant instructions for a user query
- For tasks in which lots of independent sets of instructions are needed to handle different cases, it can be beneficial to first classify the type of query and to use that classification to determine which instructions are needed.
  - Achieved by defining fixed categories and hardcoding instructions that are relevant for handling tasks in each category.

Iterative Design Process and Best Practices for Creating Prompts

4. Give GPTs time to "think"

- Instruct the model to work out its own solution before rushing to a conclusion
- Better results obtained when we explicitly instruct the model to reason from first principles before concluding.



Iterative Design Process and Best Practices for Creating Prompts

5. Use external tools

- A model can leverage external sources of information if provided as part of its input. This can help the model to generate more informed and up-to-date responses.

- Security risks relevant when using certain tools and precautions should be taken in any application that seeks to use them.

Iterative Design Process and Best Practices for Creating Prompts

6. Test changes systematically

- ***Evaluate model outputs with reference to gold-standard answers***
- Evaluation outputs can be done by model-based evaluations when there is a range of possible outputs of equal quality.

Basics of Creating Embeddings in Python – Introduction to Embeddings

- In this section you will obtain knowledge in the:
  - Embeddings and why are they important.
  - Common use cases for embedding applications.

What are Embeddings?

Embeddings: representing words or phrases as numerical vectors in a high-dimensional space

Capture the semantic and syntactic relationships between words

Importance: allow for more efficient and accurate natural language processing tasks.

Retrieval Augmented Generation (RAG)

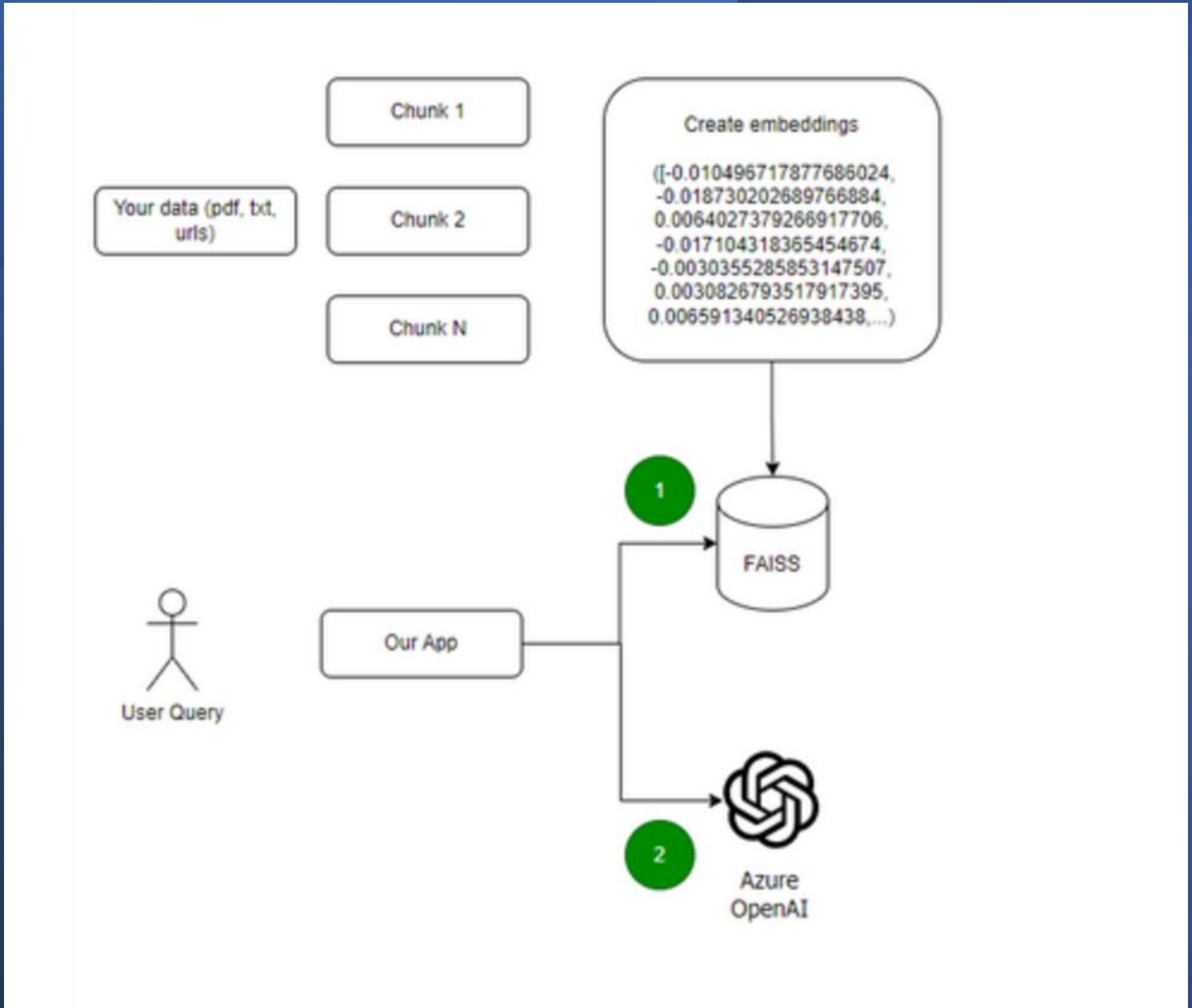


Image created by: Denise Schlesinger

Use Cases for Embeddings

Search: results are ordered by their relevance to a search term.

Clustering: text strings are categorized based on their similarity.

Recommendations: making suggestions for items with similar text strings.

Anomaly detection: where anomalies with low correlation are detected.

Diversity measurement: where the distributions of similarities are examined.

Classification: where labels are assigned to text strings based on their closest similarity match.

Conclusion

- Azure OpenAI Playground
- Basics of Prompt Engineering
- Azure OpenAI Embeddings

Feedback

- https://forms.office.com/r/VsQzxza9NN