Azure OpenAI includes several types of model:

* **GPT-4 models** are the latest generation of *generative pretrained* (GPT) models that can generate natural language and code completions based on natural language prompts.
* **GPT 3.5 models** can generate natural language and code completions based on natural language prompts. In particular, **GPT-35-turbo** models are optimized for chat-based interactions and work well in most generative AI scenarios.
* **Embeddings models** convert text into numeric vectors, and are useful in language analytics scenarios such as comparing text sources for similarities.
* **DALL-E models** are used to generate images based on natural language prompts. Currently, DALL-E models are in preview. DALL-E models aren't listed in the Azure OpenAI Studio interface and don't need to be explicitly deployed.

 **Deploying via Azure OpenAI Studio:**

* Access the Deployments page in Azure OpenAI Studio.
* Choose a model from the menu and create a new deployment.
* View information about all deployments, including status, model version, and creation date.

 **Deploying via Azure CLI:**

* Use the Azure CLI and replace variables with your resource values.

 **Deploying via REST API:**

* Utilize the REST API to deploy a model.
* In the request body, specify the base model to deploy.

**Prompt Types:**

* **Classifying content:**
  + *Prompt Example:* Tweet: "I enjoyed the trip."
  + *Completion Example:* Sentiment: Positive
* **Generating new content:**
  + *Prompt Example:* List ways of traveling
  + *Completion Example:* 1. Bike 2. Car ...
* **Holding a conversation:**
  + *Prompt Example:* A friendly AI assistant
  + *Completion Example:* See examples
* **Transformation (translation and symbol conversion):**
  + *Prompt Example:* English: Hello
  + *Completion Example:* French: bonjour
* **Summarizing content:**
  + *Prompt Example:* Provide a summary of the content {text}
  + *Completion Example:* The content shares methods of machine learning.
* **Picking up where you left off:**
  + *Prompt Example:* One way to grow tomatoes is to plant seeds.
  + *Completion Example:* (Continued from the prompt)
* **Giving factual responses:**
  + *Prompt Example:* How many moons does Earth have?
  + *Completion Example:* One

**Completion Quality:**

* The quality of completions depends on:
  + Prompt engineering
  + Model parameters
  + Data the model is trained on

**Making Calls:**

* You can make calls to your deployed model via:
  + REST API
  + Python
  + C#
  + Azure OpenAI Studio (for models with GPT-3.5 or GPT-4 base)

Playground – for model testing.

**Completions Playground Parameters:**

* **Temperature:** Controls randomness. Lowering it produces more deterministic responses, while increasing it results in more creative ones.
* **Max Length (Tokens):** Limits the number of tokens per model response.
* **Stop Sequences:** Specifies up to four sequences where the model will stop generating further tokens in a response.
* **Top Probabilities (Top P):** Controls randomness similarly to temperature but uses a different method.
* **Frequency Penalty:** Reduces the chance of repeating a token based on its frequency in the text.
* **Presence Penalty:** Reduces the chance of repeating any token that has appeared in the text.
* **Pre-Response Text:** Inserted before the model’s response.
* **Post-Response Text:** Inserted after the model’s generated response.

**Chat Playground Parameters:**

* **Max Response:** Sets a limit on the number of tokens per model response.
* **Top P:** Controls randomness similarly to temperature but uses a different method.
* **Past Messages Included:** Selects the number of past messages to include in each new API request.
* **Current Token Count:** Viewable from the Chat playground, monitors the token count to ensure it doesn't exceed the limit.

**Integrating Azure OpenAI into Your App:**

1. **Create an Azure OpenAI Resource:**
   * Access the Azure portal.
   * Search for "Azure OpenAI" and select it.
   * Click on "Create" to create the resource.
   * Fill in the required fields and choose the appropriate region.
   * Once created, you'll receive keys and an endpoint for your app.
2. **Choose and Deploy a Model:**
   * Azure OpenAI offers different families of models, including GPT, Code, and Embeddings.
   * Decide which model family suits your task best.
   * Deploy the chosen model through Azure OpenAI Studio's Deployments page.
3. **Authentication and Specification of Deployed Model:**
   * Specify the resource endpoint, key, and deployment name in your app to send requests to the deployed model.
   * This allows you to deploy multiple models within the same resource and direct requests to the appropriate model based on the task.
4. **Prompt Engineering:**
   * The input prompt significantly influences the model's response.
   * Provide detailed prompts to receive more specific and insightful responses.
   * Experiment with different prompt structures to achieve desired responses.
5. **Available Endpoints:**
   * Azure OpenAI provides several endpoints for interacting with deployed models, including Completion, ChatCompletion, and Embeddings.
   * Choose the appropriate endpoint based on your application's requirements and the capabilities of the deployed model.
   * For chat-based interactions, consider using the ChatCompletion endpoint for more realistic conversations.
6. **Endpoint Usage:**
   * Use the Completion endpoint for generating predicted completions based on input prompts.
   * Utilize the ChatCompletion endpoint for chat-based interactions, providing conversational history and role definitions.
   * The choice of endpoint depends on the model generation and the nature of the application.

Top of Form

Bottom of Form

Response quality from AI models in Azure OpenAI depends on the quality of the prompt provided. Improving prompt quality through various techniques is called prompt engineering.

**Understanding Prompt Engineering:**

1. **What is Prompt Engineering?**
   * Prompt engineering involves designing and optimizing prompts to effectively utilize AI models.
   * Well-constructed prompts significantly enhance the performance of AI models by providing context and structure for generating responses.
   * Relevant, specific, unambiguous, and well-structured prompts lead to more accurate and insightful model responses.
2. **Benefits of Prompt Engineering:**
   * Improves model performance on specific tasks by providing context and guidance.
   * Mitigates bias and enhances fairness by designing diverse and inclusive prompts.
   * Helps interpret model responses by understanding the references and logic used by the model.
3. **Methods of Prompt Engineering:**
   * Providing clear instructions and contextual content.
   * Including cues or few-shot examples to guide the model.
   * Correctly ordering content in the prompt to facilitate understanding.
4. **Considerations for API Endpoints:**
   * ChatCompletion endpoint is commonly used for chat-based interactions but can be adapted for non-chat scenarios.
   * Completion endpoint is suitable for earlier models like gpt-3, while ChatCompletion is preferred for gpt-35-turbo and later models.
5. **Adjusting Model Parameters:**
   * Parameters like temperature and top\_p (top\_probability) control randomness in the model.
   * Higher values result in more creative but less focused responses, while lower values lead to more consistent and concrete responses.
   * Experimenting with parameter adjustments can provide insights into the impact on response quality.

**Writing More Effective Prompts:**

1. **Provide Clear Instructions:**
   * Clearly state what you want from the model, providing detailed instructions.
   * Specific prompts lead to more accurate and relevant responses.
2. **Format of Instructions:**
   * Repeat important instructions at the end to mitigate recency bias.
   * Use section markers like "---" or "###" to differentiate instructions from content.
3. **Primary, Supporting, and Grounding Content:**
   * Include primary content, the main subject of the query, within prompts.
   * Provide supporting content, additional details that may influence the response.
   * Utilize grounding content, reliable and current information for accurate responses.
4. **Cues:**
   * Use leading words or cues to guide the model's response in the desired direction.
   * Cues help shape responses, especially in scenarios like code generation or summarization.

**Providing Context to Improve Accuracy:**

1. **Request Output Composition:**
   * Specify the desired structure of the output to guide the model in generating responses.
   * Formats can include Markdown tables, JSON structures, SQL queries, or other custom formats.
2. **System Message:**
   * Include a system message at the beginning of the prompt to provide instructions, perspective, or formatting guidelines to the model.
   * System messages define the context of the conversation and influence the model's response.
3. **Conversation History:**
   * Provide conversation history to enable the model to continue responding in a similar tone or style.
   * Conversation history can be from actual chat history or user-defined example conversations (few shot learning).
4. **Break Down Complex Tasks:**
   * Divide complex prompts into multiple queries to help the model better understand each individual part.
   * Breaking down prompts improves accuracy and allows for chaining responses to previous queries.
5. **Chain of Thought:**
   * Ask the model to explain its chain of thought, providing a step-by-step approach to its response.
   * Understanding the model's reasoning helps refine prompts and improve response quality.

 Azure OpenAI models have the capability to generate code from natural language prompts.

 Tasks can vary from simple one-line commands to complete applications.

 Previous generations of GPT models included specific models trained for code generation, known as codex models.

 Newer generations, like gpt-35-turbo and gpt-4, have integrated code generation into their base models.

 These newer models are proficient in both natural language understanding and code generation tasks.

 The evolution of models has eliminated the need for separate code-focused models.

 gpt-35-turbo is an example of a model that can handle both natural language and code tasks effectively.

The Azure OpenAI service enables you to use large language models (LLMs) to generate content based on natural language prompts. One of these models is the DALL-E image generation model, which is capable of creating original graphical content based on natural language descriptions of a desired image

**What is DALL-E?**

DALL-E is a neural network based model that can generate graphical data from natural language input. Put more simply, you can provide DALL-E with a description and it can generate an appropriate image.

For example, you might submit the following natural language prompt to DALL-E:

*A squirrel on a motorcycle*

This prompt could result in the generation of graphical output such as the following image:



The images generated by DALL-E are original; they are not retrieved from a curated image catalog. In other words, DALL-E is not a search system for *finding* appropriate images - it is an artificial intelligence (AI) model that *generates* new images based on the data on which it was trained.

To experiment with DALL-E, you can provision an Azure OpenAI Service resource in an Azure subscription that has been approved for access to the service, and use the DALL-E playground to submit prompts and view the resulting generated images.

When using the playground, you can adjust the settings to specify:

* The resolution (size) of the generated images. Available sizes are 256x256, 512x512, 1024x1024 (which is the default value), or 1024x1792.
* The image style to be generated (such as vivid or natural).
* The image quality (choose from standard or hd).

You can use the Azure OpenAI service REST API to consume DALL-E models from applications.

To make a REST call to the service, you need the endpoint and authorization key for the Azure OpenAI Service resource you have provisioned in Azure. You initiate the image generation process by submitting a POST request to the service endpoint with the authorization key in the header. The request must contain the following parameters in a JSON body:

prompt: The description of the image to be generated.

n: The number of images to be generated.

size: The resolution of the image(s) to be generated (256x256, 512x512, or 1024x1024).

 RAG (Retrieval Augmented Generation) with Azure OpenAI allows developers to use AI chat models that reference specific sources of information to ground responses.

 It connects pretrained models to your own data sources, utilizing Azure AI Search for search capabilities.

 The process involves receiving a user prompt, determining relevant content, querying the search index, inserting search result chunks into the prompt, and sending the prompt to Azure OpenAI.

 By default, it encourages the model to respond using your data but can be configured to use pretrained knowledge.

 RAG simplifies interaction with AI models compared to fine-tuning, which is costly and time-intensive.

 RAG still uses the stateless API, eliminating the need for custom model training with your data.  Adding your own data source in Azure OpenAI Studio's Chat playground allows you to augment prompts sent to the model.

 Data can be uploaded, sourced from a blob storage account, or connected to an existing AI Search index.

 Supported file types include .md, .txt, .html, .pdf, Microsoft Word, and PowerPoint files.

 Text extraction quality from files containing graphics or images impacts response quality.

 Using Azure OpenAI Studio to create the search resource and index is recommended for better chunking and response quality.

 For large text files or forms, using data preparation scripts improves model accuracy.

 Enabling semantic search may increase the cost but improves search result quality and citations.

 To connect data, navigate to the Chat playground, select "Add your data," and follow prompts to set up the connection and index.

 Mapping accurate fields, especially for Content data, is crucial for better responses.

 Custom indexes not created through Azure OpenAI Studio require specifying column mappings for accurate responses.

When utilizing RAG with Azure OpenAI on your own data, you have two primary methods: using the Chat playground in Azure OpenAI Studio or utilizing the API directly. Here's how to chat with the model using your own data:

**Using the Chat Playground:**

1. Navigate to the Chat playground in Azure OpenAI Studio.
2. Access the "Add your data" tab in the Assistant setup pane.
3. Connect your data source by following the prompts and specifying the necessary details.
4. Craft your prompt, ensuring it aligns with the token considerations and recommended settings.
5. Engage in a conversation with the model, leveraging your own data for more informed responses.

**Using the API:**

1. Specify the data source where your data is stored (e.g., Azure Cognitive Search).
2. Prepare your request body, including the necessary parameters such as endpoint, key, and indexName for your AI Search resource.
3. Ensure your prompt adheres to token limitations, considering question and conversation history length.
4. Send your request to the designated endpoint, ensuring it includes the Content-Type and api-key headers.