**1**          **Develop generative AI apps in Azure**

**1.1 - Plan and prepare to develop AI solutions on Azure**

AI solutions are built on machine learning *models* that encapsulate semantic relationships found in huge quantities of data; enabling applications to appear to interpret input in various formats, reason over the input data, and generate appropriate responses and predictions.

Common AI capabilities that developers can integrate into a software application include:

**Generative AI**

The ability to generate original responses to natural language *prompts*. For example, software for a real estate business might be used to automatically generate property descriptions and advertising copy for a property listing.

**Agents**

Generative AI applications that can respond to user input or assess situations autonomously, and take appropriate actions. For example, an "executive assistant" agent could provide details about the location of a meeting on your calendar, or even attach a map or automate the booking of a taxi or rideshare service to help you get there.

**Computer vision**

The ability to accept, interpret, and process visual input from images, videos, and live camera streams. For example, an automated checkout in a grocery store might use computer vision to identify which products a customer has in their shopping basket, eliminating the need to scan a barcode or manually enter the product and quantity.

**Speech**

The ability to recognize and synthesize speech. For example, a digital assistant might enable users to ask questions or provide audible instructions by speaking into a microphone, and generate spoken output to provide answers or confirmations.

**Natural language processing**

The ability to process natural language in written or spoken form, analyze it, identify key points, and generate summaries or categorizations. For example, a marketing application might analyze social media messages that mention a particular company, translate them to a specific language, and categorize them as positive or negative based on sentiment analysis.

**Information extraction**

The ability to use computer vision, speech, and natural language processing to extract key information from documents, forms, images, recordings, and other kinds of content. For example, an automated expense claims processing application might extract purchase dates, individual line item details, and total costs from a scanned receipt.

**Decision support**

The ability to use historic data and learned correlations to make predictions that support business decision making. For example, analyzing demographic and economic factors in a city to predict real estate market trends that inform property pricing decisions.

*Generative AI* represents the latest advance in artificial intelligence, and deserves some extra attention. Generative AI uses *language models* to respond to natural language *prompts*, enabling you to build conversational apps and agents that support research, content creation, and task automation in ways that were previously unimaginable.

**Azure AI services**

Microsoft Azure provides a wide range of cloud services that you can use to develop, deploy, and manage an AI solution. The most obvious starting point for considering AI development on Azure is Azure AI services; a set of out-of-the-box prebuilt APIs and models that you can integrate into your applications. see [Available Azure AI services](https://learn.microsoft.com/en-us/azure/ai-services/what-are-ai-services#available-azure-ai-services?azure-portal=true)

**Considerations for Azure AI services resources**

To use Azure AI services, you create one or more Azure AI resources in an Azure subscription and implement code in client applications to consume them. In some cases, AI services include web-based visual interfaces that you can use to configure and test your resources.

You can provision Azure AI services resources in the Azure portal (or by using BICEP or ARM templates or the Azure command-line interface) and build applications that use them directly through various service-specific APIs and SDKs.

it's better to provision Azure AI services resources as part of an *Azure AI Foundry* project - enabling you to centralize access control and cost management, and making it easier to manage shared resources and build the next generation of generative AI apps and agents.

**Single service or multi-service resource**

Most Azure AI services, such as **Azure AI Vision**, **Azure AI Language**, and so on, can be **provisioned as standalone resources, enabling you to create only the Azure resources you specifically need**. Additionally, standalone Azure AI services often include a free-tier SKU with limited functionality, enabling you to evaluate and develop with the service at no cost. Each standalone Azure AI resource provides an endpoint and authorization keys that you can use to access it securely from a client application.

Alternatively, you can provision a **multi-service resource that encapsulates multiple AI services in a single Azure resource**. Using a multi-service resource can make it easier to manage applications that use multiple AI capabilities. There are two multi-service resource types you can use:

**Azure AI services**

**Azure AI Foundry**

**Regional availability**

Some services and models are available in only a subset of Azure regions. Consider service availability and any regional quota restrictions for your subscription when provisioning Azure AI services. Use the [product availability table](https://azure.microsoft.com/explore/global-infrastructure/products-by-region/table) to check regional availability of Azure services. Use the [model availability table](https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/models#model-summary-table-and-region-availability?azure-portal=true) in the Azure OpenAI documentation to determine regional availability for Azure OpenAI models.

**Cost**

Azure AI services are charged based on usage, with different pricing schemes available depending on the specific services being used. As you plan an AI solution on Azure, use the [Azure AI services pricing](https://azure.microsoft.com/pricing/details/cognitive-services) documentation to understand pricing for the AI services you intend to incorporate into your application. You can use the [Azure pricing calculator](https://azure.microsoft.com/pricing/calculator) to estimate the costs your expected usage will incur.

**Azure AI Foundry**

Azure AI Foundry is a platform for AI development on Microsoft Azure. While you *can* provision individual Azure AI services resources and build applications that consume them without it, the project organization, resource management, and AI development capabilities of Azure AI Foundry makes it the recommended way to build all but the most simple solutions.

Azure AI Foundry provides the *Azure AI Foundry portal*, a web-based visual interface for working with AI projects. It also provides the *Azure AI Foundry SDK*, which you can use to build AI solutions programmatically.

**Azure AI Foundry projects**

In Azure AI Foundry, you manage the resource connections, data, code, and other elements of the AI solution in *projects*. There are two kinds of project:

**Foundry projects**

*Foundry projects* are associated with an **Azure AI Foundry** resource in an Azure subscription. Foundry projects provide support for Azure AI Foundry models (including OpenAI models), Azure AI Foundry Agent Service, Azure AI services, and tools for evaluation and responsible AI development.

An Azure AI Foundry resource supports the most common AI development tasks to develop generative AI chat apps and agents. In most cases, using a Foundry project provides the right level of resource centralization and capabilities with a minimal amount of administrative resource management. You can use Azure AI Foundry portal to work in projects that are based in Azure AI Foundry resources, making it easy to add connected resources and manage model and agent deployments.

**Hub-based projects**

*Hub-based projects* are associated with an **Azure AI hub** resource in an Azure subscription. Hub-based projects include an Azure AI Foundry resource, as well as managed compute, support for Prompt Flow development, and connected **Azure storage** and **Azure key vault** resources for secure data storage.

Azure AI hub resources support advanced AI development scenarios, like developing Prompt Flow based applications or fine-tuning models. You can also use Azure AI hub resources in both Azure AI Foundry portal and Azure Machine learning portal, making it easier to work on collaborative projects that involve data scientists and machine learning specialists as well as developers and AI software engineers

AI Foundry projects can be based on an *Azure AI Foundry* resource, which provides access to AI models (including Azure OpenAI), Azure AI services, and other resources for developing AI agents and chat solutions. Alternatively, projects can be based on *AI hub* resources; which include connections to Azure resources for secure storage, compute, and specialized tools.

**Azure AI Foundry based projects are great for developers who want to manage resources for AI agent or chat app development.**

**AI hub based projects are more suitable for enterprise development teams working on complex AI solutions.**

**Responsible AI**

               Core principles for responsible AI

                              Fairness

                              Reliability and safety

                              Privacy and security

                              Inclusiveness

                              Transparency

                              Accountability

**1.2**  **- Choose and deploy models from the model catalog in Azure AI Foundry portal**

**Explore the model catalog**

The *model catalog* in Azure AI Foundry provides a central repository of models that you can browse to find the right language model for your particular generative AI use case.

Selecting a foundation model for your generative AI app is important as it affects how well your app works. To find the best model for your app, you can use a structured approach by asking yourself the following questions:

* Can AI *solve* my use case?
* How do I *select* the best model for my use case?
* Can I *scale* for real-world workloads?

**Some of the options you need to consider when searching for suitable models**

               Choose between large and small language models

               Focus on a modality, task, or tool

               Specialize with regional and domain-specific models

              Balance flexibility and performance with open versus proprietary models

**How do I *select* the best model for my use case?**

To select the best language model for you use case, you need to decide on what criteria you're using to filter the models. The criteria are the necessary characteristics you identify for a model. Four characteristics you can consider are:

* **Task type**: What type of task do you need the model to perform? Does it include the understanding of only text, or also audio, or video, or multiple modalities?
* **Precision**: Is the base model good enough or do you need a fine-tuned model that is trained on a specific skill or dataset?
* **Openness**: Do you want to be able to fine-tune the model yourself?
* **Deployment**: Do you want to deploy the model locally, on a serverless endpoint, or do you want to manage the deployment infrastructure?

**Filter models for precision**

In generative AI, precision refers to the accuracy of the model in generating correct and relevant outputs. It measures the proportion of true positive results (correct outputs) among all generated outputs. High precision means fewer irrelevant or incorrect results, making the model more reliable.

When integrating a language model into an app, you can choose between a base model or a fine-tuned model. **A base model, like GPT-4, is pretrained on a large dataset and can handle various tasks but can lack precision** for specific domains. Techniques like prompt engineering can improve this, but sometimes fine-tuning is necessary.

**A fine-tuned model is trained further on a smaller, task-specific dataset to improve its precision and ability** to generate relevant outputs for specific applications. You can either use a fine-tuned model or fine-tune a model yourself.

**Filter models for performance**

You can evaluate your model performance at different phases, using various evaluation approaches.

When you're exploring models through the Azure AI Foundry model catalog, you can use **model benchmarks** to compare publicly available metrics like coherence and accuracy across models and datasets. These benchmarks can help you in the initial exploration phase, but give little information on how the model would perform in your specific use case.

**Can I *scale* for real-world workloads?**

You selected a model for your use case and have successfully built a prototype. Now, you need to understand how to scale for real-world workloads.

Considerations for scaling a generative AI solution include:

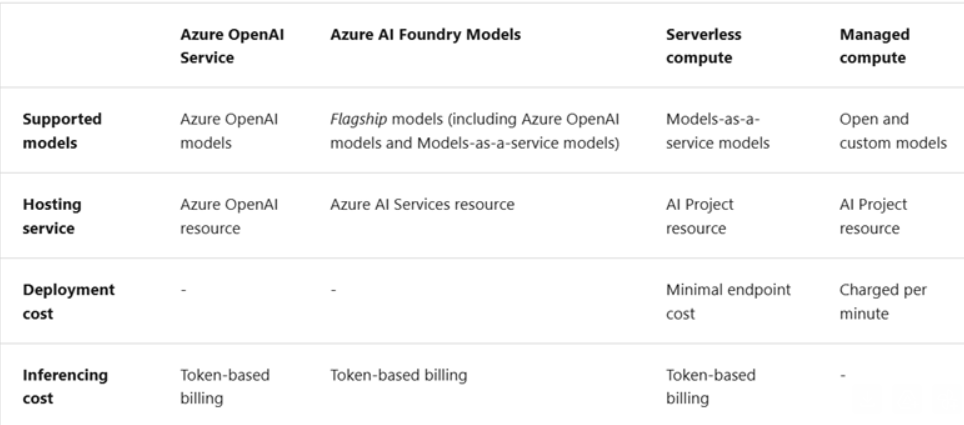
* **Model deployment**: Where will you deploy the model for the best balance of performance and cost?
* **Model monitoring and optimization**: How will you monitor, evaluate, and optimize model performance?
* **Prompt management**: How will you orchestrate and optimize prompts to maximize the accuracy and relevance of generated responses?
* **Model lifecycle**: How will you manage model, data, and code updates as part of an ongoing *Generative AI Operations* (GenAIOps) lifecycle?

**Deploy a model to an endpoint**

When you develop a generative AI app, you need to integrate language models into your application. To be able to use a language model, you need to deploy the model.

When you deploy a language model from the model catalog with the Azure AI Foundry, you get an endpoint, which consists of a **target URI** (Uniform Resource Identifier) and a unique **key**.

**Deploy a language model with Azure AI Foundry**



**Optimize model performance**

After you deploy your model to an endpoint, you can start interacting with it to see how it works. Let's explore how you can use prompt engineering techniques to optimize your model's performance:

               Apply prompt patterns to optimize your model's output

               Instruct the model to act as a persona

               Ask for better question suggestions

               Specify the desired format for responses

               Ask for an explanation of reasoning

               Add context

               Apply model optimization strategies

**1.3 -**  **Develop an AI app with the Azure AI Foundry SDK**

**Azure AI Foundry SDK**

Azure AI Foundry SDK is a set of packages and services designed to work together to enable developers to write code that uses resources in an Azure AI Foundry project. With the Azure AI Foundry SDK, developers can create applications that connect to a project, access the resource connections and models in that project, and use them to perform AI operations, such as sending prompts to a generative AI model and processing the responses

The SDK provides Python and Microsoft C# .NET libraries that you can use to build AI applications based on Azure AI Foundry projects.

**pip install azure-ai-projects**

**pip install azure-identity**

**Using the SDK to connect to a project**

The first task in most Azure AI Foundry SDK code is to connect to an Azure AI Foundry project. Each project has a unique *endpoint*, which you can find on the project's **Overview** page in the Azure AI Foundry portal.

The project provides multiple endpoints and keys, including:

* An endpoint for the project itself; which can be used to access project connections, agents, and models in the Azure AI Foundry resource.
* An endpoint for Azure OpenAI Service APIs in the project's Azure AI Foundry resource.
* An endpoint for Azure AI services APIs (such as Azure AI Vision and Azure AI Language) in the Azure AI Foundry resource.

You can use the project endpoint in your code to create an **AIProjectClient** object, which provides a programmatic proxy for the project.

**Work with project connections**

Each Azure AI Foundry project includes **connected resources**, which are defined both at the *parent* (Azure AI Foundry resource or hub) level, and at the *project* level. Each resource is a *connection* to an external service, such as Azure storage, Azure AI Search, Azure OpenAI, or another Azure AI Foundry resource.

With the Azure AI Foundry SDK, you can connect to a project and retrieve connections; which you can then use to consume the connected services.

The **AIProjectClient** object in Python has a **connections** property, which you can use to access the resource connections in the project. Methods of the **connections** object include:

* connections.list(): Returns a collection of connection objects, each representing a connection in the project. You can filter the results by specifying an optional **connection\_type** parameter with a valid enumeration, such as ConnectionType.AZURE\_OPEN\_AI.
* connections.get(connection\_name, include\_credentials): Returns a connection object for the connection with the name specified. If the **include\_credentials** parameter is **True** (the default value), the credentials required to connect to the connection are returned - for example, in the form of an API key for an Azure AI services resource.



**Create a chat client**

The specific libraries and code used to build a chat client depends on how the target model has been deployed in the Azure AI Foundry project. You can deploy models to the following model hosting solutions:

* **Azure AI Foundry Models**: A single endpoint for multiple models of different types, including OpenAI models and others from the Azure AI Foundry model catalog. Models are consumed through an **Azure AI Foundry** resource connection in the project (either the default **Azure AI Foundry** resource for the project or another resource connection that has been added to the project).
* **Azure OpenAI**: A single endpoint for OpenAI models hosted in Azure. Models are consumed through an **Azure OpenAI** resource connection in the project.
* **Serverless API**: A model-as-a-service solution in which each deployed model is accessed through a unique endpoint and hosted in the Azure AI Foundry project.
* **Managed compute**: A model-as-a-service solution in which each deployed model is accessed through a unique endpoint hosted in custom compute.

**Building a client app for Azure AI Foundry Models**

When you have deployed models to the Azure AI model inference service, you can use the Azure AI Foundry SDK to write code that create a **ChatCompletionsClient** object, which you can then use to chat with a deployed model. One of the benefits of using this model deployment type is that you can easily switch between deployed models by changing one parameter in your code (the model deployment name), making it a great way to test against multiple models while developing an app.



The **ChatCompletionsClient** class uses **Azure AI Inference** library. Use  
**pip install azure-ai-inference**

**Using the Azure OpenAI SDK**

In the Azure AI Foundry SDK for Python, the **AIProjectClient** class provides a **get\_azure\_openai\_client()** method that you can use to create an Azure OpenAI client object. You can then use the classes and methods defined in the Azure OpenAI SDK to consume an OpenAI model deployed to Azure Foundry Models.

The following Python code sample uses the Azure AI Foundry and Azure OpenAI SDKs to chat with a model deployment named **gpt-4o-model**.



**1.4 - Get started with prompt flow to develop language model apps in the Azure AI Foundry**

**Understand the development lifecycle of a large language model (LLM) app**

The lifecycle consists of the following stages:

* **Initialization**: Define the use case and design the solution.
* **Experimentation**: Develop a flow and test with a small dataset.
* **Evaluation and refinement**: Assess the flow with a larger dataset.
* **Production**: Deploy and monitor the flow and application.

During both evaluation and refinement, and production, you might find that your solution needs to be improved. You can revert back to experimentation during which you develop your flow continuously, until you're satisfied with the results.

**Initialization**

During initialization you:

* Define the **objective**
* Collect a **sample dataset**
* Build a **basic prompt**
* Design the **flow**

When collecting or creating the sample dataset, you should ensure diversity in the data to cover various scenarios and edge cases. You should also remove any privacy sensitive information from the dataset to avoid any vulnerabilities.

**Experimentation**

You collected a sample dataset, and decided on which categories you want to be classified into. You designed a flow that takes a data as input, and uses an LLM to classify. To test whether your flow generates the expected output, you run it against your sample dataset.

The experimentation phase is an iterative process during which you (1) **run** the flow against a sample dataset. You then (2) **evaluate** the prompt's performance. If you're (3) satisfied with the result, you can **move on** to evaluation and refinement. If you think there's room for improvement, you can (4) **modify** the flow by changing the prompt or flow itself.

**Evaluation and refinement**

When you're satisfied with the output of the flow that classifies data, based on the sample dataset, you can assess the flow's performance against a larger dataset.

By testing the flow on a larger dataset, you can evaluate how well the LLM application generalizes to new data. During evaluation, you can identify potential bottlenecks or areas for optimization or refinement.

When you edit your flow, you should first run it against a smaller dataset before running it again against a larger dataset. Testing your flow with a smaller dataset allows you to more quickly respond to any issues.

Once your LLM application appears to be robust and reliable in handling various scenarios, you can decide to move the LLM application to production.

**Production**

During production you:

* **Optimize** the flow that classifies incoming articles for efficiency and effectiveness.
* **Deploy** your flow to an endpoint. When you call the endpoint, the flow is triggered to run and the desired output is generated.
* **Monitor** the performance of your solution by collecting usage data and end-user feedback. By understanding how the application performs, you can improve the flow whenever necessary.

**Understand core components and explore flow types**

**Understand a flow**

Prompt flow is a feature within Azure AI Foundry that allows you to author **flows**. Flows are executable workflows often consist of three parts:

1. **Inputs**: Represent data passed into the flow. Can be different data types like strings, integers, or boolean.
2. **Nodes**: Represent *tools* that perform data processing, task execution, or algorithmic operations.
3. **Outputs**: Represent the data produced by the flow.

## Explore the tools available in prompt flow

Three common tools are:

* **LLM tool**: Enables custom prompt creation utilizing Large Language Models.
* **Python tool**: Allows the execution of custom Python scripts.
* **Prompt tool**: Prepares prompts as strings for complex scenarios or integration with other tools.

Each tool is an executable unit with a specific function. You can use a tool to perform tasks like summarizing text, or making an API call. You can use multiple tools within one flow and use a tool multiple times.

Whenever you add a new node to your flow, adding a new tool, you can define the expected inputs and outputs. A node can use one of the whole flow's inputs, or another node's output, effectively linking nodes together.

By defining the inputs, connecting nodes, and defining the desired outputs, you can create a flow. Flows help you create LLM applications for various purposes.

**Understand the types of flows**

There are three different types of flows you can create with prompt flow:

* **Standard flow**: Ideal for general LLM-based application development, offering a range of versatile tools.
* **Chat flow**: Designed for conversational applications, with enhanced support for chat-related functionalities.
* **Evaluation flow**: Focused on performance evaluation, allowing the analysis and improvement of models or applications through feedback on previous runs.

# Explore connections and runtimes

When you create a Large Language Model (LLM) application with prompt flow, you first need to configure any necessary **connections** and **runtimes**.

# Explore connections

# Whenever you want your flow to connect to external data source, service, or API, you need your flow to be authorized to communicate with that external service. When you create a connection, you configure a secure link between prompt flow and external services, ensuring seamless and safe data communication.

Depending on the type of connection you create, the connection securely stores the endpoint, API key, or credentials necessary for prompt flow to communicate with the external service. Any necessary secrets aren't exposed to users, but instead are stored in an Azure Key Vault.

By setting up connections, users can easily reuse external services necessary for tools in their flows.

Prompt flow connections play pivotal roles in two scenarios. They automate API credential management, simplifying and securing the handling of sensitive access information. Additionally, they enable secure data transfer from various sources, crucial for maintaining data integrity and privacy across different environments.

**Explore runtimes**

After creating your flow, and configuring the necessary connections your tools use, you want to run your flow. To run the flow, you need compute, which is offered through prompt flow **runtimes**.

Runtimes are a combination of a **compute instance** providing the necessary compute resources, and an **environment** specifying the necessary packages and libraries that need to be installed before being able to run the flow.

When you use runtimes, you have a controlled environment where flows can be run and validated, ensuring that everything works as intended in a stable setting. A default environment is available for quick development and testing. When you require other packages to be installed, you can [create a custom environment](https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/how-to-customize-environment-runtime).

# Explore variants and monitoring options

During production, you want to optimize and deploy your flow. Finally, you want to monitor your flows to understand when improving your flows is necessary.

You can optimize your flow by using variants, you can deploy your flow to an endpoint, and you can monitor your flow by evaluating key metrics.

**Explore variants**

Prompt flow **variants** are versions of a tool node with distinct settings. Currently, variants are only supported in the LLM tool, where a variant can represent a different prompt content or connection setting. Variants allow users to customize their approach for specific tasks, like, summarizing news articles.

Some benefits of using variants are:

* **Enhance the quality of your LLM generation**: Creating diverse variants of an LLM node helps find the best prompt and settings for high-quality content.
* **Save time and effort**: Variants allow for easy management and comparison of different prompt versions, streamlining historical tracking and reducing the effort in prompt tuning.
* **Boost productivity**: They simplify the optimization of LLM nodes, enabling quicker creation and management of variations, leading to better results in less time.
* **Facilitate easy comparison**: Variants enable side-by-side result comparisons, aiding in choosing the most effective variant based on data-driven decisions.

**Deploy your flow to an endpoint**

When you're satisfied with the performance of your flow, you can choose to deploy it to an **online endpoint**. Endpoints are URLs that you can call from any application. When you make an API call to an online endpoint, you can expect (almost) immediate response.

When you deploy your flow to an online endpoint, prompt flow generates a URL and key so you can safely integrate your flow with other applications or business processes. When you invoke the endpoint, a flow is run and the output is returned in real-time. As a result, deploying flows to endpoints can for example generate chat or agentic responses that you want to return in another application.

**Monitor evaluation metrics**

In prompt flow, monitoring evaluation metrics is key to understanding your LLM application's performance, ensuring they meet real-world expectations and deliver accurate results.

To understand whether your application is meeting practical needs, you can collect end-user feedback and assess the application's usefulness. Another approach to understanding whether your application is performing well, is by comparing LLM predictions with expected or *ground truth* responses to gauge accuracy and relevance. Evaluating the LLM's predictions is crucial for keeping LLM applications reliable and effective.

**Metrics**

The key metrics used for monitoring evaluation in prompt flow each offer unique insight into the performance of LLMs:

* **Groundedness**: Measures alignment of the LLM application's output with the input source or database.
* **Relevance**: Assesses how pertinent the LLM application's output is to the given input.
* **Coherence**: Evaluates the logical flow and readability of the LLM application's text.
* **Fluency**: Assesses the grammatical and linguistic accuracy of the LLM application's output.
* **Similarity**: Quantifies the contextual and semantic match between the LLM application's output and the ground truth.

Metrics like *groundedness*, *relevance*, *coherence*, *fluency*, and *similarity* are key for quality assurance, ensuring that interactions with your LLM applications are accurate and effective. Whenever your LLM application doesn't perform as expected, you need to revert back to experimentation to iteratively explore how to improve your flow.

**1.5 - Develop a RAG-based solution with your own data using Azure AI Foundry**

**Ungrounded prompts and responses**

When you use a language model to generate a response to a prompt, the only information that the model has to base the answer on comes from the data on which it was trained - which is often just a large volume of uncontextualized text from the Internet or some other source.

The result will likely be a grammatically coherent and logical response to the prompt, but because it isn't grounded in relevant, factual data, it's uncontextualized; and may in fact be inaccurate and include "invented" information.

**Grounded prompts and responses**

In contrast, you can use a data source to *ground* the prompt with some relevant, factual context. The prompt can then be submitted to a language model, including the grounding data, to generate a contextualized, relevant, and accurate response.

The data source can be any repository of relevant data. For example, you could use data from a product catalog database to ground the prompt "Which product should I use to do *X*?" so that the response includes relevant details of products that exist in the catalog.

**Understand how to ground your language model**

Language models excel in generating engaging text, and are ideal as the base for agents. Agents provide users with an intuitive chat-based application to receive assistance in their work. When designing an agent for a specific use case, you want to ensure your language model is grounded and uses factual information that is relevant to what the user needs.

Though language models are trained on a vast amount of data, they may not have access to the knowledge you want to make available to your users. To ensure that an agent is grounded on specific data to provide accurate and domain-specific responses, you can use Retrieval Augmented Generation (RAG).

**Understanding RAG**

RAG is a technique that you can use to ground a language model. In other words, it's a process for retrieving information that is relevant to the user's initial prompt. In general terms, the RAG pattern incorporates the following steps:

1. **Retrieve** grounding data based on the initial user-entered prompt.
2. **Augment** the prompt with grounding data.
3. Use a language model to **generate** a grounded response.

By retrieving context from a specified data source, you ensure that the language model uses relevant information when responding, instead of relying on its training data.

Using RAG is a powerful and easy-to-use technique for many cases in which you want to ground your language model and improve the factual accuracy of your generative AI app's responses.

**Adding grounding data to an Azure AI project**

You can use Azure AI Foundry to build a custom age that uses your own data to ground prompts. Azure AI Foundry supports a range of data connections that you can use to add data to a project, including:

Azure Blob Storage

Azure Data Lake Storage Gen2

Microsoft OneLake

You can also upload files or folders to the storage used by your AI Foundry project.

**Make your data searchable**

When you want to create an agent that uses your own data to generate accurate answers, you need to be able to search your data efficiently. When you build an agent with the Azure AI Foundry, you can use the integration with **Azure AI Search** to retrieve the relevant context in your chat flow.

**Azure AI Search is a retriever** that you can include when building a language model application with prompt flow. Azure AI Search allows you to bring your own data, index your data, and query the index to retrieve any information you need.

**Using a *vector* index**

While a text-based index will improve search efficiency, you can usually achieve a better data retrieval solution by using a *vector*-based index that contains *embeddings* that represent the text tokens in your data source.

An embedding is a special format of data representation that a search engine can use to easily find the relevant information. More specifically, an embedding is a vector of floating-point numbers.

By representing words and their meanings with vectors, you can extract relevant context from your data source even when your data is stored in different formats (text or image) and languages.

When you want to be able to use vector search to search your data, you need to create embeddings when creating your search index. To create embeddings for your search index, you can use an Azure OpenAI embedding model available in Azure AI Foundry.

**Creating a search index**

In Azure AI Search, a search index describes how your content is organized to make it searchable. Imagine a library containing many books. You want to be able to search through the library and retrieve the relevant book easily and efficiently. To make the library searchable, you create a catalog that contains any relevant data about books to make any book easy to find. A library’s catalog serves as the search index.

Though there are different approaches to creating an index, the integration of Azure AI Search in Azure AI Foundry makes it easy for you to create an index that is suitable for language models. You can add your data to Azure AI Foundry, after which you can use Azure AI Search to create an index in the Azure AI Foundry portal using an embedding model. The index asset is stored in Azure AI Search and queried by Azure AI Foundry when used in a chat flow.

How you configure your search index depends on the data you have and the context you want your language model to use. For example, **keyword search enables you to retrieve information that exactly matches the search query**. **Semantic search already takes it one step further by retrieving information that matches the meaning** of the query instead of the exact keyword, using semantic models. Currently, the **most advanced technique is vector search**, which creates embeddings to represent your data.

**Searching an index**

**Keyword search**: Identifies relevant documents or passages based on specific keywords or terms provided as input.

**Semantic search**: Retrieves documents or passages by understanding the meaning of the query and matching it with semantically related content rather than relying solely on exact keyword matches.

**Vector search**: Uses mathematical representations of text (vectors) to find similar documents or passages based on their semantic meaning or context.

**Hybrid search**: Combines any or all of the other search techniques. Queries are executed in parallel and are returned in a unified result set.

When you create a search index in Azure AI Foundry, you're guided to configuring an index that is most suitable to use in combination with a language model. When your search results are used in a generative AI application, hybrid search gives the most accurate results.

Hybrid search is a combination of keyword (and full text), and vector search, to which semantic ranking is optionally added. When you create an index that is compatible with hybrid search, the retrieved information is precise when exact matches are available (using keywords), and still relevant when only conceptually similar information can be found (using vector search).

**1.6 Fine-tune a language model with Azure AI Foundry**

**Understand when to fine-tune a language model**

Prompt engineering is a quick and easy way to improve how the model acts, and what the model needs to know. When you want to improve the quality of the model even further, there are two common techniques that are used:

Retrieval Augmented Generation (RAG): Ground your data by first retrieving context from a data source before generating a response.  
Fine-tuning: Train a base language model on a dataset before integrating it in your application.

RAG is most commonly applied when you need the model's responses to be factual and grounded in specific data.  
On the other hand, when you want the model to behave a certain way, fine-tuning can help you achieve your goal. You can also use a combination of optimization strategies, like RAG and a fine-tuned model, to improve your language application.  
  
To maximize the consistency of the model's behavior, you can fine-tune a base model with your own training data.

**Prepare your data to fine-tune a chat completion model**

Fine-tuning involves combining a suitable foundation model to use as a base, and with a set of training data that includes example prompts and responses that the model can learn from.

The quality of the dataset has a large effect on the quality of your model. Though you need less data than when you would train a language model from scratch, you still might need enough data to maximize the consistency of your desired model's behavior. How much data you need depends on your use case.

When you fine-tune a language model for chat completion, the data you use to fine-tune a model is a collection of sample conversations. More specifically, the data should contain three components:

The system message

The user message

The assistant's response

{"messages": [{"role": "system", "content": "You are an Xbox customer support agent whose primary goal is to help users with issues they are experiencing with their Xbox devices. You are friendly and concise. You only provide factual answers to queries, and do not provide answers that are not related to Xbox."}, {"role": "user", "content": "Is Xbox better than PlayStation?"}, {"role": "assistant", "content": "I apologize, but I cannot provide personal opinions. My primary job is to assist you with any issues related to your Xbox device. Do you have any Xbox-related issues that need addressing?"}]}

A few things to keep in mind when you use real data is to:

Remove any personal or sensitive information.

Not only focus on creating a large training dataset, but also ensure your dataset includes a diverse set of examples.

When preparing your dataset to fine-tune a language model, you should understand your desired model behaviors, create a dataset in JSONL format, and ensure the examples you include are high quality and diverse. By preparing your dataset, you have a higher chance that the fine-tuned model improves your chat application's performance.

**Explore fine-tuning language models in Azure AI Foundry portal**

When you want to use a fine-tuned model to generate responses in a chat application, you need to use a base model that can be fine-tuned on a chat completion task. The Azure AI Foundry model catalog allows you to filter based on fine-tuning tasks to decide which base model to select. You can, for example, select a GPT-4 or Llama-2-7b model to fine-tune on your own training data.

**Select the base model**

You can filter the available models based on the task you want to fine-tune a model for. Per task, you have several options for foundation models to choose from. When deciding between foundation models for a task, you can examine the description of the model, and the referenced model card.

Some considerations you can take into account when deciding on a foundation model before fine-tuning are:

Model capabilities: Evaluate the capabilities of the foundation model and how well they align with your task. For example, a model like BERT is better at understanding short texts.

Pretraining data: Consider the dataset used for pretraining the foundation model. For example, GPT-2 is trained on unfiltered content from the internet that can result in biases.

Limitations and biases: Be aware of any limitations or biases that might be present in the foundation model.

Language support: Explore which models offer the specific language support or multilingual capabilities that you need for your use case.

**Configure the fine-tuning job**

To configure a fine-tuning job using the Azure AI Foundry portal, you need to do the following steps:

Select a base model.

Select your training data.

(Optional) Select your validation data.

Configure the advanced options.

When you submit a model for fine-tuning, the model is further trained on your data. To configure the fine-tuning or training job, you can specify the following advanced options:

**batch\_size** - The batch size to use for training. The batch size is the number of training examples used to train a single forward and backward pass. In general, larger batch sizes tend to work better for larger datasets. The default value and the maximum value for this property are specific to a base model. A larger batch size means that model parameters are updated less frequently, but with lower variance.

**learning\_rate\_multiplier** - The learning rate multiplier to use for training. The fine-tuning learning rate is the original learning rate used for pretraining multiplied by this value. Larger learning rates tend to perform better with larger batch sizes. We recommend experimenting with values in the range 0.02 to 0.2 to see what produces the best results. A smaller learning rate can be useful to avoid overfitting.

**n\_epochs** - The number of epochs to train the model for. An epoch refers to one full cycle through the training dataset.

**seed** - The seed controls the reproducibility of the job. Passing in the same seed and job parameters should produce the same results, but can differ in rare cases. If a seed isn't specified, one is generated for you.

You can always deploy a fine-tuned model. After deploying the model, you can test it to assess its performance. When you're satisfied with your fine-tuned model, you can integrate the deployed model with your chat application.

**1.7 Implement a responsible generative AI solution in Azure AI Foundry**

**Plan a responsible generative AI solution**

The Microsoft guidance for responsible generative AI is designed to be practical and actionable. It defines a four stage process to develop and implement a plan for responsible AI when using generative models. The four stages in the process are:

**Map** - potential harms that are relevant to your planned solution.

**Measure** - the presence of these harms in the outputs generated by your solution.

**Mitigate** - the harms at multiple layers in your solution to minimize their presence and impact, and ensure transparent communication about potential risks to users.

**Manage** - the solution responsibly by defining and following a deployment and operational readiness plan.

**Map potential harms**

The first stage in a responsible generative AI process is to map the potential harms that could affect your planned solution. There are four steps in this stage

Identify potential harms

Prioritize identified harms

Test and verify the prioritized harms

Document and share the verified harms

Red teaming is a strategy that is often used to find security vulnerabilities or other weaknesses that can compromise the integrity of a software solution. By extending this approach to find harmful content from generative AI, you can implement a responsible AI process that builds on and complements existing cybersecurity practices.

**Measure potential harms**

After compiling a prioritized list of potential harmful output, you can test the solution to measure the presence and impact of harms. Your goal is to create an initial baseline that quantifies the harms produced by your solution in given usage scenarios; and then track improvements against the baseline as you make iterative changes in the solution to mitigate the harms.

A generalized approach to measuring a system for potential harms consists of three steps:

Prepare a diverse selection of input prompts that are likely to result in each

potential harm that you have documented for the system.

Submit the prompts to the system and retrieve the generated output.

Apply pre-defined criteria to evaluate the output and categorize it according to the level of potential harm it contains. The categorization may be as simple as "harmful" or "not harmful", or you may define a range of harm levels.

Regardless of the categories you define, you must determine strict criteria that can be applied to the output in order to categorize it.

**Mitigate potential harms**

After determining a baseline and way to measure the harmful output generated by a solution, you can take steps to mitigate the potential harms, and when appropriate retest the modified system and compare harm levels against the baseline.

Mitigation of potential harms in a generative AI solution involves a layered approach, in which mitigation techniques can be applied at each of four layers:

Model

Safety System

System message and grounding

User experience

**The model layer**

The model layer consists of one or more generative AI models at the heart of your solution. For example, your solution may be built around a model such as GPT-4.

Mitigations you can apply at the model layer include:

Selecting a model that is appropriate for the intended solution use. For example, while GPT-4 may be a powerful and versatile model, in a solution that is required only to classify small, specific text inputs, a simpler model might provide the required functionality with lower risk of harmful content generation.

Fine-tuning a foundational model with your own training data so that the responses it generates are more likely to be relevant and scoped to your solution scenario.

**The safety system layer**

The safety system layer includes platform-level configurations and capabilities that help mitigate harm. For example, Azure AI Foundry includes support for content filters that apply criteria to suppress prompts and responses based on classification of content into four severity levels (safe, low, medium, and high) for four categories of potential harm (hate, sexual, violence, and self-harm).

Other safety system layer mitigations can include abuse detection algorithms to determine if the solution is being systematically abused (for example through high volumes of automated requests from a bot) and alert notifications that enable a fast response to potential system abuse or harmful behavior.

**The system message and grounding layer**

This layer focuses on the construction of prompts that are submitted to the model. Harm mitigation techniques that you can apply at this layer include:

Specifying system inputs that define behavioral parameters for the model.

Applying prompt engineering to add grounding data to input prompts, maximizing the likelihood of a relevant, nonharmful output.

Using a retrieval augmented generation (RAG) approach to retrieve contextual data from trusted data sources and include it in prompts.

**The user experience layer**

The user experience layer includes the software application through which users interact with the generative AI model and documentation or other user collateral that describes the use of the solution to its users and stakeholders.

Designing the application user interface to constrain inputs to specific subjects or types, or applying input and output validation can mitigate the risk of potentially harmful responses.

Documentation and other descriptions of a generative AI solution should be appropriately transparent about the capabilities and limitations of the system, the models on which it's based, and any potential harms that may not always be addressed by the mitigation measures you have put in place.

**Manage a responsible generative AI solution**

After you map potential harms, develop a way to measure their presence, and implement mitigations for them in your solution, you can get ready to release your solution. Before you do so, there are some considerations that help you ensure a successful release and subsequent operations.

**Complete prerelease reviews**

Before releasing a generative AI solution, identify the various compliance requirements in your organization and industry and ensure the appropriate teams are given the opportunity to review the system and its documentation. Common compliance reviews include:

Legal

Privacy

Security

Accessibility

**Release and operate the solution**

A successful release requires some planning and preparation. Consider the following guidelines:

Devise a phased delivery plan that enables you to release the solution initially to restricted group of users. This approach enables you to gather feedback and identify problems before releasing to a wider audience.

Create an incident response plan that includes estimates of the time taken to respond to unanticipated incidents.

Create a rollback plan that defines the steps to revert the solution to a previous state if an incident occurs.

Implement the capability to immediately block harmful system responses when they're discovered.

Implement a capability to block specific users, applications, or client IP addresses in the event of system misuse.

Implement a way for users to provide feedback and report issues. In particular, enable users to report generated content as "inaccurate", "incomplete", "harmful", "offensive", or otherwise problematic.

Track telemetry data that enables you to determine user satisfaction and identify functional gaps or usability challenges. Telemetry collected should comply with privacy laws and your own organization's policies and commitments to user privacy.

**Utilize Azure AI Foundry Content Safety**

Several Azure AI resources provide built-in analysis of the content they work with, including Language, Vision, and Azure OpenAI by using content filters.

Azure AI Foundry Content Safety provides more features focusing on keeping AI and copilots safe from risk. These features include detecting inappropriate or offensive language, both from input or generated, and detecting risky or inappropriate inputs.

**Features in Foundry Content Safety include:**

Prompt shields - Scans for the risk of user input attacks on language models

Groundedness detection - Detects if text responses are grounded in a user's source content

Protected material detection - Scans for known copyrighted content

Custom categories - Define custom categories for any new or emerging patterns

**1.8 Evaluate generative AI performance in Azure AI Foundry portal**

**Assess the model performance**

When you develop a generative AI app, you use a language model in your chat application to generate a response.

To help you decide which model you want to integrate into your application, you can evaluate the performance of an individual language model:

An input is provided to a language model, and a response is generated as output. The model is then evaluated by analyzing the input, the output, and optionally comparing it to predefined expected output.

**Model benchmarks**

Model benchmarks are publicly available metrics across models and datasets. These benchmarks help you understand how your model performs relative to others. Some commonly used benchmarks include:

Accuracy: Compares model generated text with correct answer according to the dataset. Result is one if generated text matches the answer exactly, and zero otherwise.

Coherence: Measures whether the model output flows smoothly, reads naturally, and resembles human-like language

Fluency: Assesses how well the generated text adheres to grammatical rules, syntactic structures, and appropriate usage of vocabulary, resulting in linguistically correct and natural-sounding responses.

GPT similarity: Quantifies the semantic similarity between a ground truth sentence (or document) and the prediction sentence generated by an AI model.

**Manual evaluations**

Manual evaluations involve human raters who assess the quality of the model's responses. This approach provides insights into aspects that automated metrics might miss, such as context relevance and user satisfaction. Human evaluators can rate responses based on criteria like relevance, informativeness, and engagement.

**AI-assisted metrics**

AI-assisted metrics use advanced techniques to evaluate model performance. These metrics can include:

**Generation quality metrics**: These metrics evaluate the overall quality of the generated text, considering factors like creativity, coherence, and adherence to the desired style or tone.

**Risk and safety metrics**: These metrics assess the potential risks and safety concerns associated with the model's outputs. They help ensure that the model doesn't generate harmful or biased content.

**Natural language processing metrics**

Natural language processing (NLP) metrics are also valuable in evaluating model performance. One such metric is the F1-score, which measures the ratio of the number of shared words between the generated and ground truth answers. The F1-score is useful for tasks like text classification and information retrieval, where precision and recall are important. Other common NLP metrics include:

BLEU: Bilingual Evaluation Understudy metric

METEOR: Metric for Evaluation of Translation with Explicit Ordering

ROUGE: Recall-Oriented Understudy for Gisting Evaluation

**Manually evaluate the performance of a model**

During the early phases of the development of your generative AI app, you want to experiment and iterate quickly. To easily assess whether your selected language model and app, created with prompt flow, meet your requirements, you can manually evaluate models and flows in the Azure AI Foundry portal.

**Prepare your test prompts**

To begin the manual evaluation process, it's essential to prepare a diverse set of test prompts that reflect the range of queries and tasks your app is expected to handle. These prompts should cover various scenarios, including common user questions, edge cases, and potential failure points. By doing so, you can comprehensively assess the app's performance and identify areas for improvement.

**Test the selected model in the chat playground**

When you develop a chat application, you use a language model to generate a response. You create a chat application by developing a prompt flow that encapsulates your chat application's logic, which can use multiple language models to ultimately generate a response to a user question.

Before you test your app's response, you can test the selected language model's response to verify the individual model works as expected. You can test a model you deployed in the Azure AI Foundry portal by interacting with it in the chat playground.

**Evaluate multiple prompts with manual evaluations**

The chat playground is an easy way to get started. When you want to manually evaluate multiple prompts more quickly, you can use the manual evaluations feature. This feature allows you to upload a dataset with multiple questions, and optionally add an expected response, to evaluate the model's performance on a larger test dataset.

**Automated evaluations**

Automated evaluations in Azure AI Foundry portal enable you to assess the quality and content safety performance of models, datasets, or prompt flows.

**Evaluation data**

To evaluate a model, you need a dataset of prompts and responses (and optionally, expected responses as "ground truth"). You can compile this dataset manually or use the output from an existing application; but a useful way to get started is to use an AI model to generate a set of prompts and responses related to a specific subject. You can then edit the generated prompts and responses to reflect your desired output, and use them as ground truth to evaluate the responses from another model.

**Evaluation metrics**

Automated evaluation enables you to choose which evaluators you want to assess your model's responses, and which metrics those evaluators should calculate. There are evaluators that help you measure:

**AI Quality**: The quality of your model's responses is measured by using AI models to evaluate them for metrics like coherence and relevance and by using standard NLP metrics like F1 score, BLEU, METEOR, and ROUGE based on ground truth (in the form of expected response text)

**Risk and safety**: evaluators that assess the responses for content safety issues, including violence, hate, sexual content, and content related to self-harm.