# Plan and prepare to develop AI solutions on Azure

## Learning objectives

Microsoft Azure offers multiple services that enable developers to build amazing AI-powered solutions. Proper planning and preparation involves identifying the services you'll use and creating an optimal working environment for your development team.

By the end of this module, you'll be able to:

- Identify common AI capabilities that you can implement in applications
- Describe **Azure AI Services and considerations** for using them
- Describe **Azure AI Foundry and considerations** for using it
- Identify appropriate **developer tools and SDKs** for an AI project
- Describe considerations for **responsible AI**

## Introduction

The growth in the use of artificial intelligence (AI) in general, and generative AI in particular means that developers are increasingly required to create comprehensive AI solutions. These solutions need to combine **machine learning models, AI services, prompt engineering solutions, and custom code**.

Microsoft Azure provides multiple services that you can use to create AI solutions. However, before embarking on an AI application development project, it's useful to consider the available options for services, tools, and frameworks as well as some principles and practices that can help you succeed.

This module explores some of the key considerations for planning an AI development project, and introduces **Azure AI Foundry**; a comprehensive platform for AI development on Microsoft Azure.

# What is AI?

The term **"Artificial Intelligence" (AI)** covers a wide range of software capabilities that enable applications to exhibit human-like behavior. AI has been around for many years, and its definition has varied as the technology and use cases associated with it have evolved. In today's technological landscape, 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:

| Capability | Description |
|---|---|
| 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**. |

| Capability | Description |
| --- | --- |
| 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. |

Determining the specific AI capabilities you want to include in your application can help you identify the most appropriate AI services that you'll need to provision, configure, and use in your solution.

## A closer look at generative AI

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.

The language models used in generative AI solutions can be **large language models (LLMs)** that have been trained on huge volumes of data and include many millions of parameters; or they can be **small language models (SLMs)** that are optimized for specific scenarios with lower overhead. Language models commonly respond to text-based prompts with natural language text; though increasingly new **multi-modal models** are able to handle image or speech prompts and respond by generating text, code, speech, or images.

# 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. The following table lists some commonly used Azure AI services (for a full list of all available Azure AI services, see Available Azure AI services).

| Service | Description |
| --- | --- |
| Azure OpenAI | Azure OpenAI in Foundry Models provides access to OpenAI generative AI models including the **GPT family of large and small language models** and **DALL-E image-generation models** within a scalable and securable cloud service on Azure. |
| Azure AI Vision | The Azure AI Vision service provides a set of models and APIs that you can use to implement common computer vision functionality in an application. With the AI Vision service, you can **detect common objects in images, generate captions, descriptions, and tags based on image contents, and read text in images**. |
| Azure AI Speech | The Azure AI Speech service provides APIs that you can use to implement **text to speech** and **speech to text transformation**, as well as specialized speech-based capabilities like **speaker recognition and translation**. |
| Azure AI Language | The Azure AI Language service provides models and APIs that you can use to analyze natural language text and perform tasks such as **entity extraction, sentiment analysis, and summarization**. The AI Language service also provides functionality to help you **build conversational language models and question answering solutions**. |
| Azure AI Content Safety | Azure AI Content Safety provides developers with access to advanced algorithms for processing images and text and **flagging content that is potentially offensive, risky, or otherwise undesirable**. |
| Azure AI Translator | The Azure AI Translator service uses state-of-the-art language models to **translate text** between a large number of languages. |
| Azure AI Face | The Azure AI Face service is a specialist computer vision implementation that can **detect, analyze, and recognize human faces**. Because of the potential risks associated with personal identification and misuse of this capability, access to some features of the AI Face service are restricted to approved customers. |
| Azure AI Custom Vision | The Azure AI Custom Vision service enables you to train and use custom computer vision models for **image classification and object detection**. |
| Azure AI Document Intelligence | With Azure AI Document Intelligence, you can use pre-built or custom models to **extract fields from complex documents such as invoices, receipts, and forms**. |

| Service | Description |
| --- | --- |
| Azure AI Content Understanding | The Azure AI Content Understanding service provides **multi-modal content analysis** capabilities that enable you to build models to extract data from forms and documents, images, videos, and audio streams. |
| Azure AI Search | The Azure AI Search service uses a pipeline of AI skills based on other Azure AI Services and custom code to **extract information from content and create a searchable index**. AI Search is commonly used to **create vector indexes for data** that can then be used to ground prompts submitted to generative AI language models, such as those provided in the Azure OpenAI service. |

## 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 - for example to train a custom image classification model using the Custom Vision service you can use the visual interface to upload training images, manage training jobs, and deploy the resulting model.

> Note: 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**. However, as we'll discuss later in this module, in most medium to large-scale development scenarios 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:

### 1. Azure AI services

The Azure AI Services resource type includes the following services, making them available from a single endpoint:

- Azure AI Speech
- Azure AI Language
- Azure AI Translator
- Azure AI Vision
- Azure AI Face
- Azure AI Custom Vision
- Azure AI Document Intelligence

### 2. Azure AI Foundry

The Azure AI Foundry resource type includes the following services, and supports working with them through an Azure AI Foundry project:

- Azure OpenAI
- Azure AI Speech
- Azure AI Language
- Azure AI Foundry Content Safety
- Azure AI Translator
- Azure AI Vision
- Azure AI Face
- Azure AI Document Intelligence
- Azure AI Content Understanding

Using a multi-service resource can make it easier to manage applications that use multiple AI capabilities.

## 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 to check regional availability of Azure services. Use the model availability table in the Azure OpenAI service 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 documentation to understand pricing for the AI services you intend to incorporate into your application. You can use the Azure 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:

### 1. 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.

### 2. 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

> Tip: For more information about Azure AI Foundry project types, see What is Azure AI Foundry?.

> Note: the notes below are from the old course.

## Hubs and projects

In Azure AI Foundry, you manage the resources, assets, code, and other elements of the AI solution in hubs and projects. **Hubs provide a top-level container for managing shared resources, data, connections and security configuration for AI application development**. A hub can support multiple projects, in which developers collaborate on building a specific solution.

### Hubs

A hub provides a centrally managed collection of shared resources and management configuration for AI solution development. You need at least one hub to use all of the solution development features and capabilities of AI Foundry.

In a hub, you can define shared resources to be used across multiple projects. When you create a hub using the Azure AI Foundry portal, an **Azure AI Hub** resource is created in a resource group associated with the hub. Additionally, the following resources are created for the hub:

- A multi-service **Azure AI services** resource to provide access to Azure OpenAI and other Azure AI services.
- A **Key vault** in which sensitive data such as connections and credentials can be stored securely.
- A **Storage account** for data used in the hub and its projects.
- Optionally, an **Azure AI Search** resource that can be used to index data and support grounding for generative AI prompts.

You can create more resources as required (for example, an **Azure AI Face** resource) and add it to the hub (or an individual project) by defining a connected resource. As you create more items in your hub, such as compute instances or endpoints, more resources will be created for them in the Azure resource group.

Access to the resources in a hub is governed by creating *users* and assigning them to *roles*. An IT administrator can manage access to the resources centrally at the hub level, and projects associated with the hub inherit the resources and role assignments; enabling development teams to use the resources they need without needing to request access on a project-by-project basis.

## Projects

A hub can support one or more projects, each of which is used to organize the resources and assets required for a particular AI development effort.

Users can collaborate in a project, sharing data in project-specific storage containers and connected resources, and using the shared resources defined in the hub associated with the project. Azure AI Foundry provides tools and functionality within a project that developers can use to build AI solutions efficiently, including:

- A **model catalog** in which you can find and deploy machine learning models from multiple sources, including Azure OpenAI and the Hugging Face model library.
- **Playgrounds** in which you can test prompts with generative AI models.
- Access to **Azure AI services**, including visual interfaces to experiment with and configure services as well as endpoints and keys that you can use to connect to them from client applications.
- **Visual Studio Code** containers that define a hosted development environment in which you can write, test, and deploy code.
- **Fine-tuning** functionality for generative AI models that you need to customize based on custom training prompts and responses.
- **Prompt Flow**, a prompt orchestration tool that you can use to define the logic for a generative AI application's interaction with a model.
- Tools to assess, evaluate, and improve your AI applications, including *tracing, evaluations, and content safety and security management*.
- Management of project **assets**, including models and endpoints, data and indexes, and deployed web apps.

# Considerations for Azure AI Foundry

When planning an AI solution built on Azure AI Foundry, there are some additional considerations to those discussed previously in relation to Azure AI services.

## Hub and project organization

Plan your hub and project organization for the most effective management of resources and efficiency of administration. **Use Hubs to centralize management of users and shared resources that are involved in related projects, and then add project-specific resources as necessary.** For example, an organization might have separate software development teams for each area of the business, so it may make sense to create separate hubs for each business area (such as Marketing, HR, and so on) in which AI application development projects for each business area can be created. The shared resources in each hub will automatically be available in projects created in those hubs.

## Connected resources

At the hub level, an IT administrator can create shared resource connections in a hub that will be used in downstream projects. Projects access the connected resources by proxy on behalf of project users, so users in those projects don't need direct access to those resources in order to use them within the context of the project. Connections in a hub are automatically available in new projects in the hub without further requests to the IT administrator. If an individual project needs access to a specific resource that other projects in the same hub don't use, you can create more connected resources at the project level.

As you plan your Azure AI Foundry hubs and projects, identify the shared connected resources you should add to each hub so that they're inherited by projects in that hub, while allowing for project-level exceptions.

# Security and authorization

For each hub and project, identify the users who will need access and the roles to which they should be assigned.

## Hub-level roles

Hub-level roles can perform infrastructure management tasks, such as creating hub-level connected resources or new projects. The default roles in a hub are:

- **Owner**: Full access to the hub, including the ability to manage and create new hubs and assign permissions. This role is automatically assigned to the hub creator

- **Contributor**: Full access to the hub, including the ability to create new hubs, but isn't able to manage hub permissions on the existing resource.
- **Azure AI Developer**: All permissions except create new hubs and manage the hub permissions.
- **Azure AI Inference Deployment Operator**: All permissions required to create a resource deployment within a resource group.
- **Reader**: Read only access to the hub. This role is automatically assigned to all project members within the hub.

### Project-level roles

Project-level roles determine the tasks that a user can perform within an individual project. The default roles in a project are:

- **Owner**: Full access to the project, including the ability to assign permissions to project users.
- **Contributor**: Full access to the project but can't assign permissions to project users.
- **Azure AI Developer**: Permissions to perform most actions, including create deployments, but can't assign permissions to project users.
- **Azure AI Inference Deployment Operator**: Permissions to perform all actions required to create a resource deployment within a resource group.
- **Reader**: Read only access to the project.

## Regional availability

As with all Azure services, the availability of specific Azure AI Foundry capabilities can vary by region. As you plan your solution, determine regional availability for the capabilities you require.

## Costs and quotas

In addition to the cost of the Azure AI services your solution uses, there are costs associated with Azure AI Foundry related to the resources that support hubs and projects as well as storage and compute for assets, development, and deployed solutions. You should consider these costs when planning to use Azure AI Foundry for AI solution development.

In addition to service consumption costs, you should consider the resource quotas you need to support the AI applications you intend to build. Quotas are used to limit utilization, and play a key role in cost management and managing Azure capacity. In some cases, you may need to request additional quota to increase rate limits for AI model operations or available compute for development and solution deployment.

# Developer tools and SDKs

While you can perform many of the tasks needed to develop an AI solution directly in the Azure AI Foundry portal, developers also need to write, test, and deploy code.

## Development tools and environments

There are many development tools and environments available, and developers should choose one that supports the languages, SDKs, and APIs they need to work with and with which they're most comfortable. For example, a developer who focuses strongly on building applications for Windows using the .NET Framework might prefer to work in an integrated development environment (IDE) like **Microsoft Visual Studio**. Conversely, a web application developer who works with a wide range of open-source languages and libraries might prefer to use a code editor like **Visual Studio Code (VS Code)**. Both of these products are suitable for developing AI applications on Azure.

## The Azure AI Foundry for Visual Studio Code extension

When developing Azure AI Foundry based generative AI applications in Visual Studio Code, you can use the **Azure AI Foundry for Visual Studio Code extension** to simplify key tasks in the workflow, including:

- Creating a project.
- Selecting and deploying a model.
- Testing a model in the playground.
- Creating an agent.

> Tip: For more information about using the Azure AI Foundry for Visual Studio Code extension, see Work with the Azure AI Foundry for Visual Studio Code extension.

## The Azure AI Foundry VS Code container image

As an alternative to installing and configuring your own development environment, within Azure AI Foundry portal, you can create compute and use it to host a container image for VS Code (installed locally or as a hosted web application in a browser). The benefit of using the container image is that it includes the latest versions of the SDK packages you're most likely to work with when building AI applications with Azure AI Foundry.

## GitHub and GitHub Copilot

GitHub is the world's most popular platform for **source control and DevOps management**, and can be a critical element of any team development effort. Visual Studio and VS Code (including the Azure AI Foundry VS Code container image) both provide native integration with GitHub, and access to GitHub Copilot; an AI assistant that can significantly improve developer productivity and effectiveness.

> Tip: For more information about using GitHub Copilot in Visual Studio Code, see GitHub Copilot in VS Code.

## Programming languages, APIs, and SDKs

You can develop AI applications using many common programming languages and frameworks, including **Microsoft C#, Python, Node, TypeScript, Java**, and others. When building AI solutions on Azure, some common SDKs you should plan to install and use include:

- The Azure AI Foundry SDK, which enables you to write code to connect to Azure AI Foundry projects and access resource connections, which you can then work with using service-specific SDKs.
- The Azure AI Foundry Models API, which provides an interface for working with generative AI model endpoints hosted in Azure AI Foundry.
- The Azure OpenAI in Azure AI Foundry Models API, which enables you to build chat applications based on OpenAI models hosted in Azure AI Foundry.
- Azure AI Services SDKs - AI service-specific libraries for multiple programming languages and frameworks that enable you to consume Azure AI Services resources in your subscription. You can also use Azure AI Services through their REST APIs.
- The Azure AI Foundry Agent Service, which is accessed through the Azure AI Foundry SDK and can be integrated with frameworks like Semantic Kernel to build comprehensive AI agent solutions.

# Responsible AI

It's important for software engineers to consider the impact of their software on users, and society in general; including considerations for its responsible use. When the application is imbued with artificial intelligence, these considerations are particularly important due to the nature of how AI systems work and inform decisions; often based on probabilistic models, which are in turn dependent on the data with which they were trained.

The human-like nature of AI solutions is a significant benefit in making applications user-friendly, but it can also lead users to place a great deal of trust in the application's ability to make correct decisions. **The potential for harm to individuals or groups through incorrect predictions or misuse of AI capabilities is a major concern**, and software engineers building AI-enabled solutions should apply due consideration to mitigate risks and ensure fairness, reliability, and adequate protection from harm or discrimination.

Let's discuss some core principles for responsible AI that have been adopted at Microsoft.

## Fairness

**AI systems should treat all people fairly**. For example, suppose you create a machine learning model to support a loan approval application for a bank. The model should make predictions of whether or not the loan should be approved **without incorporating any bias** based on gender, ethnicity, or other factors that might result in an unfair advantage or disadvantage to specific groups of applicants.

Fairness of machine learned systems is a highly active area of ongoing research, and some software solutions exist for evaluating, quantifying, and mitigating unfairness in machine learned models. However, tooling alone isn't sufficient to ensure fairness. **Consider fairness from the beginning of the application development process; carefully reviewing training data to ensure it's representative of all potentially affected subjects, and evaluating predictive performance for subsections of your user population throughout the development lifecycle**.

## Reliability and safety

AI systems should **perform reliably and safely**. For example, consider an AI-based software system for an autonomous vehicle; or a machine learning model that diagnoses patient symptoms and recommends prescriptions. Unreliability in these kinds of system can result in substantial risk to human life.

As with any software, AI-based software application development must be subjected to rigorous testing and deployment management processes to ensure that they **work as expected** before release. Additionally, software engineers need to **take into account the probabilistic nature of machine learning models**, and apply appropriate thresholds when **evaluating confidence scores for predictions**.

## Privacy and security

AI systems should be secure and respect privacy. The machine learning models on which AI systems are based rely on large volumes of data, which may contain personal details that must be kept private. Even after models are trained and the system is in production, they use new data to make predictions or take action that may be subject to privacy or security concerns; so **appropriate safeguards to protect data and customer content** must be implemented.

## Inclusiveness

AI systems should **empower everyone and engage people**. AI should bring benefits to all parts of society, regardless of physical ability, gender, sexual orientation, ethnicity, or other factors.

One way to optimize for inclusiveness is to **ensure that the design, development, and testing of your application includes input from as diverse a group of people as possible**.

## Transparency

**AI systems should be understandable**. Users should be made fully aware of the purpose of the system, how it works, and what limitations may be expected.

For example, when an AI system is based on a machine learning model, you should generally make users aware of **factors that may affect the accuracy of its predictions**, such as the number of cases used to train the model, or the specific features that have the most influence over its predictions. You should also share information about the **confidence score** for predictions.

When an AI application relies on personal data, such as a facial recognition system that takes images of people to recognize them; you should **make it clear to the user how their data is used and retained, and who has access to it**.

## Accountability

People should be accountable for AI systems. Although many AI systems seem to operate autonomously, ultimately it's the **responsibility of the developers** who trained and validated the models they use, and defined the logic that bases decisions on model predictions to **ensure that the overall system meets responsibility requirements**. To help meet this goal, designers and developers of AI-based solution should work within a framework of governance and organizational principles that ensure the solution meets responsible and legal standards that are clearly defined.

> Tip: For more information about Microsoft's principles for responsible AI, see the Microsoft responsible AI site.

# Exercise - Prepare for an AI development project

## Prepare for an AI development project

In this exercise, you use Azure AI Foundry portal to create a project, ready to build an AI solution.

- Open Azure AI Foundry portal
- Create a project
- Review project connections
- Test a generative AI model
- Summary: In this exercise, you've explored Azure AI Foundry, and seen how to create and manage projects and their related resources.

# Module assessment

1. Which Azure resource provides language and vision services from a single endpoint? **Azure AI service**.
2. You plan to create a simple chat app that uses a generative AI model. What kind of project should you create? **Azure AI Foundry Project**.
3. Which SDK enables you to connect to resources in a project? **Azure AI Foundry SDK**.
4. How should you provide access to resources for developers who will work on multiple AI projects? **Create resource connections in an Azure AI Foundry hub**.
5. Which SDK enables you to connect to shared resources in a hub? **Azure AI Foundry SDK**.

# Summary

In this module, you explored some of the key considerations when planning and preparing for AI application development. You've also had the opportunity to become familiar with **Azure AI Foundry**, the recommended platform for developing AI solutions on Azure.

# Choose and deploy models from the model catalog in Azure AI Foundry portal

Choose the various language models that are available through the **Azure AI Foundry's model catalog**. Understand how to **select, deploy, and test a model**, and to **improve its performance**.

## Learning objectives

By the end of this module, you'll be able to:

- **Select a language model** from the model catalog.
- **Deploy a model to an endpoint**.
- **Test a model** and **improve the performance** of the model.

## Introduction

Generative AI applications are built on **language models**. The development process usually starts with an exploration and comparison of available foundation models to **find the one that best suits the particular needs of your application**. After selecting a suitable model, you **deploy it to an endpoint** where it can be consumed by a client application or AI agent.

Foundation models, such as the GPT family of models, are state-of-the-art language models designed to understand, generate, and interact with natural language. Some common use cases for models are:

- **Speech-to-text** and **text-to-speech conversion**. For example, generate subtitles for videos.
- **Machine translation**. For example, **translate text** from English to Japanese.
- **Text classification**. For example, **label an email as spam or not spam**.
- **Entity extraction**. For example, **extract keywords or names** from a document.
- **Text summarization**. For example, generate a short one-paragraph summary from a multi-page document.
- **Question answering**. For example, provide answers to questions like "What is the capital of France?"
- **Reasoning**. For example, solve a mathematical problem.

In this module, you focus on exploring **foundation models used for question answering**. The foundation models you explore can be used for chat applications in which you use a language model to generate a response to a user's question.

> Note: The latest breakthrough in generative AI models is owed to the development of the **Transformer** architecture. Transformers were introduced in the **Attention is all you need** paper by Vaswani, et al. from 2017. The Transformer architecture provided two innovations to NLP that resulted in the emergence of foundation models:
>
> - Instead of processing words sequentially, Transformers process each word independently and in parallel by using **attention**.
> - Next to the semantic similarity between words, Transformers use **positional encoding** to include the information about the position of a word in a sentence.

# 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?

Let's explore each of these questions.

## Can AI solve my use case?

Nowadays we have thousands of language models to choose from. The main challenge is to understand if there's a model that satisfies your needs and to answer the question: *Can AI solve my use case?*

**To start answering this question, you need to discover, filter, and deploy a model**. You can explore the available language models through three different catalogs:

- Hugging Face: Vast catalog of open-source models across various domains.

- GitHub: Access to diverse models via GitHub Marketplace and GitHub Copilot.
- Azure AI Foundry: Comprehensive catalog with robust tools for deployment.

Though you can use each of these catalogs to explore models, the **model catalog in Azure AI Foundry** makes it easiest to explore and deploy a model to build you prototype, while offering the best selection of models.

Let's explore some of the options you need to consider when searching for suitable models.

## Choose between large and small language models

First of all, you have a choice between **Large Language Models (LLMs)** and **Small Language Models (SLMs)**.

- **LLMs** like GPT-4, Mistral Large, Llama3 70B, Llama 405B, and Command R+ are powerful AI models designed for tasks that require **deep reasoning, complex content generation, and extensive context understanding**.
- **SLMs** like Phi3, Mistral OSS models, and Llama3 8B are efficient and cost-effective, while still handling many common Natural Language Processing (NLP) tasks. They're perfect for **running on lower-end hardware or edge devices**, where cost and speed are more important than model complexity.

## Focus on a modality, task, or tool

Language models like GPT-4 and Mistral Large are also known as **chat completion models**, designed to generate coherent and contextually appropriate text-based responses. When you need higher levels of performance in complex tasks like math, coding, science, strategy, and logistics, you can also use **reasoning models** like DeepSeek-R1 and o1.

Beyond text-based AI, some models are **multi-modal**, meaning they can *process images, audio, and other data types alongside text*. Models like `GPT-4o` and `Phi3-vision` are capable of analyzing and generating both text and images. Multi-modal models are useful when your application needs to process and understand images, such as in computer vision or document analysis. Or when you want to build an AI app that interacts with visual content, such as a digital tutor explaining images or charts.

If your use case involves **generating images**, tools like `DALL·E 3` and `Stability` AI can create realistic visuals from text prompts. **Image generation models** are great for designing marketing materials, illustrations, or digital art.

Another group of task-specific models are **embedding models** like `Ada` and `Cohere` . Embeddings models convert text into numerical representations and are used to improve search relevance by

understanding semantic meaning. These models are often implemented in **Retrieval Augmented Generation (RAG)** scenarios to enhance recommendation engines by linking similar content.

When you want to build an application that interacts with other software tools dynamically, you can add function calling and JSON support. These capabilities allow AI models to work efficiently with structured data, making them useful for automating API calls, database queries, and structured data processing.

## Specialize with regional and domain-specific models

**Certain models are designed for specific languages, regions, or industries**. These models can outperform general-purpose generative AI in their respective domains. For example:

- `Core42 JAIS` is an Arabic language LLM, making it the best choice for applications targeting Arabic-speaking users.
- `Mistral Large` has a strong focus on European languages, ensuring better linguistic accuracy for multilingual applications.
- `Nixtla TimeGEN-1` specializes in **time-series forecasting**, making it ideal for financial predictions, supply chain optimization, and demand forecasting.

If your project has regional, linguistic, or industry-specific needs, these models can provide more relevant results than general-purpose AI.

## Balance flexibility and performance with open versus proprietary models

You also need to **decide whether to use open-source models or proprietary models**, each with its own advantages.

**Proprietary models** are best for cutting-edge performance and enterprise use. Azure offers models like OpenAI's `GPT-4`, `Mistral Large`, and `Cohere Command R+`, which deliver industry-leading AI capabilities. These models are ideal for businesses needing **enterprise-level security, support, and high accuracy**.

**Open-source models** are best for flexibility and cost-efficiency. There are hundreds of open-source models available in the Azure AI Foundry model catalog from Hugging Face, and models from Meta, Databricks, Snowflake, and Nvidia. Open models give developers **more control, allowing fine-tuning, customization, and local deployment**.

Whatever model you choose, you can use the **Azure AI Foundry model catalog**. Using models through the model catalog meets the key enterprise requirements for usage:

- **Data and privacy**: you get to decide what happens with your data.

- **Security and compliance**: built-in security.
- **Responsible AI and content safety**: evaluations and content safety.

Now you know the language models that are available to you, you should have an understanding of whether AI can indeed solve your use case. If you think a language model would enrich your application, you then need to select the specific model that you want to deploy and integrate.

# 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?

You already explored the various types of models available in the previous section. Now, let's explore in more detail how precision and performance can be important filters when choosing a model.

## 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.

| Benchmark | Description |
|---|---|
| **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. |
| **Groundedness** | Measures alignment between the model's generated answers and the input data. |
| **GPT Similarity** | Quantifies the semantic similarity between a ground truth sentence (or document) and the prediction sentence generated by an AI model. |
| **Quality index** | **A comparative aggregate score between 0 and 1**, with better-performing models scoring a higher value. |
| **Cost** | The cost of using the model based on a **price-per-token**. Cost is a useful metric with which to compare quality, enabling you to determine an appropriate tradeoff for your needs. |

To evaluate how a selected model performs regarding your specific requirements, you can consider manual or automated evaluations. **Manual evaluations** allow you to rate your model's responses. **Automated evaluations** include traditional machine learning metrics and AI-assisted metrics that are calculated and generated for you.

When you evaluate a model’s performance, it's common to start with manual evaluations, as they quickly assess the quality of the model’s responses. For more systematic comparisons, automated

evaluations using metrics like **precision, recall, and F1 score** based on your own ground truth offer a faster, scalable, and more objective approach.

## 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?

Azure AI Foundry provides visual and code-first tools that can help you build and maintain a scalable generative AI solution.

# 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. Let's explore how to **deploy language models in the Azure AI Foundry**, after first understanding why to deploy a model.

## Why deploy a model?

You train a model to generate output based on some input. To get value out of your model, you need a solution that allows you to **send input to the model, which the model processes, after which the output is visualized for you**.

With generative AI apps, the most common type of solution is a **chat application** that expects a user question, which the model processes, to generate an adequate response. The response is then visualized to the user as a response to their question.

You can integrate a language model with a chat application by deploying the model to an endpoint. **An endpoint is a specific URL where a deployed model or service can be accessed.** Each model deployment typically has its own unique endpoint, which allows different applications to **communicate with the model through an API (Application Programming Interface)**.

When a user asks a question:

1. An API request is sent to the endpoint.
2. The endpoint specifies the model that processes the request.
3. The result is sent back to the app through an API response.

Now that you understand why you want to deploy a model, let's explore the deployment options with Azure AI Foundry.

## Deploy a language model with Azure AI Foundry

When you deploy a language model with Azure AI Foundry, you have several types available, which depend on the model you want to deploy.

Deploy options include:

- **Standard deployment**: Models are hosted in the Azure AI Foundry project resource.
- **Serverless compute**: Models are hosted in Microsoft-managed dedicated serverless endpoints in an Azure AI Foundry hub project.
- **Managed compute**: Models are hosted in managed virtual machine images in an Azure AI Foundry hub project.

The associated cost depends on the type of model you deploy, which deployment option you choose, and what you are doing with the model:

| | Standard deployment | Serverless compute | Managed compute |
|---|---|---|---|
| Supported models | Azure AI Foundry models (including Azure OpenAI models and Models-as-a-service models) | Foundry Models with pay-as-you-go billing | Open and custom models |
| Hosting service | Azure AI Foundry resource | AI Project resource in a hub | AI Project resource in a hub |

| | Standard deployment | Serverless compute | Managed compute |
|---|---|---|---|
| Billing basis | Token-based billing | Token-based billing | Compute-based billing |

> Note: **Standard deployment** is recommended for most scenarios.

# 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

**The quality of the questions you send to the language model, directly influences the quality of the responses you get back**. You can carefully construct your question, or prompt, to receive better and more interesting responses. **The process of designing and optimizing prompts to improve the model's performance is also known as prompt engineering**.

Prompt engineering requires users to **ask relevant, specific, unambiguous, and well-structured questions**, instructing the model to generate more accurate responses. To understand how to create well-defined prompts, let's explore some patterns that help you improve the output of a model:

- Instruct the model to act as a **persona**.
- Guide the model to **suggest better questions**.
- Provide a template to generate output in **a specific format**.
- Understand how a model reasons by asking it to reflect.
- Add context to improve the accuracy of the model's output.

> Note: The patterns listed here are based on the work by White et al., (2023) who constructed a prompt pattern catalog.

As a developer, an effective approach is to add instructions to your deployed model in the form of a **system prompt**. The system prompt sets the model's behavior and allows you to guide the model without exposing the end user to the instructions. The best results are often achieved through the assignment of **an explicit system prompt and guidance (or templates)** that help users submit prompts that reflect the patterns described here.

## Instruct the model to act as a persona

When you want the model to take a specific point of view or perspective when answering a question, you can ask it to act as a persona.

For example, you can ask the model to respond as if it's a seasoned marketing professional if you're working in sales and need insights about a new CRM software.

| | No persona specified | With persona specified |
|---|---|---|
| System prompt | None | You're a seasoned marketing professional that writes advertising copy for an audience of technical customers. |
| User prompt | Write a one-sentence description of a CRM product. | Write a one-sentence description of a CRM product. |
| Model response | A CRM product is a software tool designed to manage a company's interactions with current and potential customers, encompassing sales, marketing, customer service, and technical support. | Experience seamless customer relationship management with our CRM product, designed to streamline your business operations, enhance customer interactions, and drive sales growth with its intuitive interface and robust analytics tools. |

You can create different personas for various business roles. Think of roles like a product manager, data analyst, customer service expert, or teacher. Define the persona and add more detail to what you expect that persona to know or do to get tailored, context-driven responses.

## Ask for better question suggestions

When you want the model to help refine your question, you can **ask it to suggest better ways to phrase your query**. You can also ask it to provide extra context.

For example, if you're seeking advice on cooking a meal, you can prompt the model to suggest clarifying questions to help you get a more targeted answer.

| | No suggestions given | With suggestions given |
|---|---|---|
| System prompt | You're an AI chef who can advise on menus. | You're an AI chef who can advise on menus. You're approachable and always want to provide the best advice you can. |

| | No suggestions given | With suggestions given |
|---|---|---|
| User prompt | I have to host a dinner party for four people. What should I cook? | I have to host a dinner party for four people. What should I cook? **What other information do you need** to help me plan a great meal for my guests? |

When you encourage the model to suggest clarifications or extra details can help you ask the right questions and achieve better, more accurate answers in fewer interactions.

## Specify the desired format for responses

When you want the model to generate output in a specific format, you can provide a template or structure in your prompt.

For example, if you're a sports reporting composing a historical article, you can request that the model follow **a specific template, which includes headings, bullet points, and data breakdowns**.

| | No template specified | With template specified |
|---|---|---|
| System prompt | You're a helpful AI assistant. | You're a helpful AI assistant for sports reporters. |
| User prompt | What happened in the 2018 Soccer World Cup final? | What happened in the 2018 Soccer World Cup final? Format the result to show the match date, location, and the two teams competing. Then the final score, and finally any notable events that occurred during the match. |

You can apply this approach to other scenarios where a specific format is needed, such as generating emails, summaries, proposals, or even code and scripts. Define the format template clearly and provide details on how you want the output structured to get consistent and organized responses.

You can also use a **one-shot** or **few-shots approach** by providing one or more examples to help the model identify a desired pattern.

## Ask for an explanation of reasoning

When you want the model to explain the reasoning behind its answers, you can ask the model to automatically reflect on its rationale and assumptions after providing a response.

For example, if you're working on a mathematical problem, you can ask the model to explain the reasoning behind specific calculations.

| | No reflection specified | With reflection specified |
|---|---|---|
| System prompt | You're an AI math assistant. | You're an AI math assistant. **You always explain your answers**. |
| User prompt | A right-angled triangle has a hypotenuse side of length 3 cm and an adjacent side length of 2 cm. What is the length of the remaining side? | A right-angled triangle has a hypotenuse side of length 3 cm and an adjacent side length of 2 cm. What is the length of the remaining side? |

You can apply this approach when you want explanations in data analysis, marketing strategy, or technical troubleshooting. When you ask the model to define its reasoning, you use a technique called **chain-of-thought** to make it think step by step.

### Add context

When you want the model to focus on specific topics, you can specify the context to consider. You can also **tell the model to ignore irrelevant information**.

For example, if you're planning a trip, you can provide the model with more context to help improve the relevance of its response.

| | No context specified | With context specified |
|---|---|---|
| System prompt | You're an AI travel assistant. | You're an AI travel assistant. |
| User question | When should I visit Edinburgh? | When should I visit Edinburgh? I'm particularly interested in attending Scotland's home matches in the Six Nations rugby tournament. |

By defining what the model should focus on or disregard, you can ensure the conversation stays on track and generate more relevant, tailored responses.

You can specify the context by **describing what it should or shouldn't include, and by connecting the model to data sources it should retrieve context from before generating an answer**.

## Apply model optimization strategies

> Note: This section discusses options and considerations for model optimization that you may consider beyond prompt engineering. A full exploration of how to apply these optimization

> strategies is beyond the scope of this module.

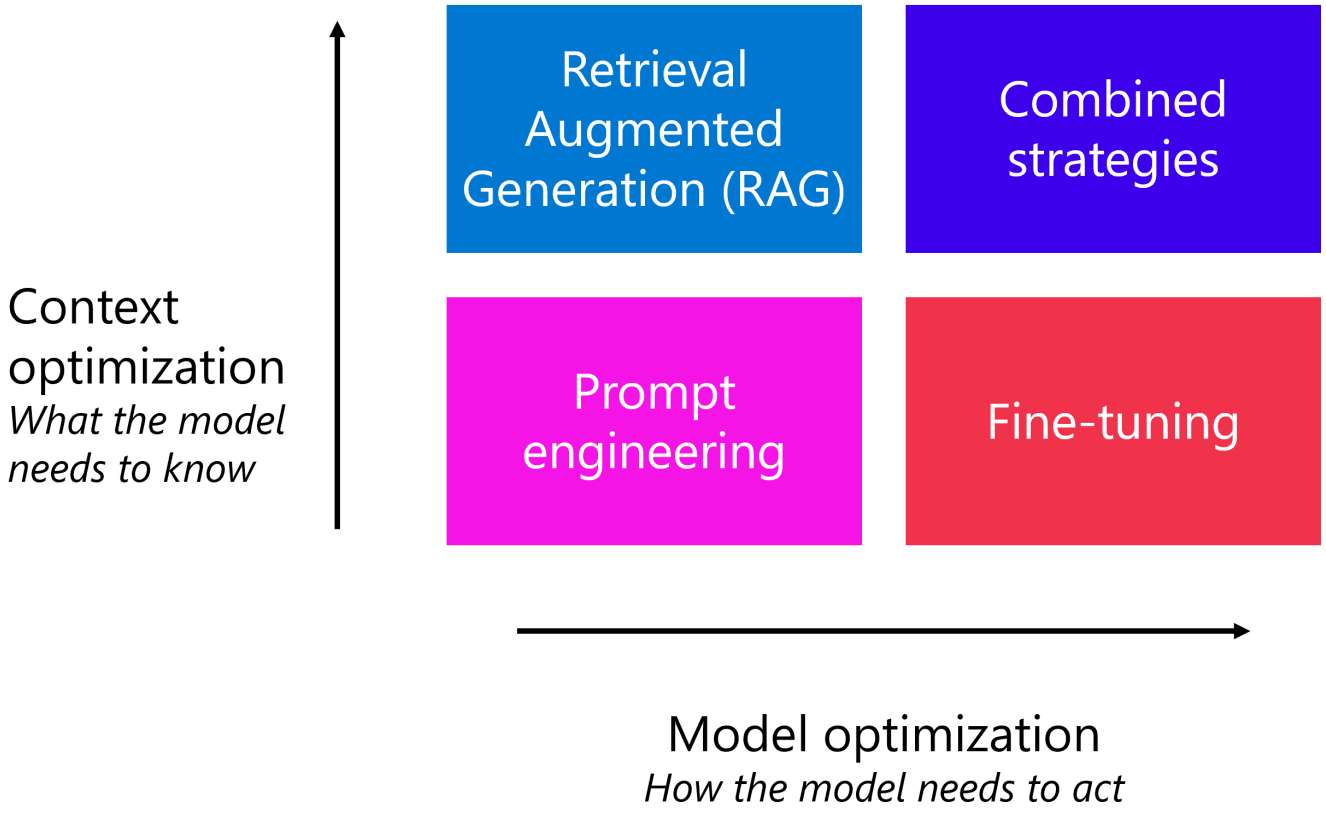
Prompt engineering can be an effective way to optimize model responses, but in some cases it may not provide sufficient context or guidance to always meet your exact needs. As a developer, you can consider the following additional optimization strategies to improve the relevance of your generative AI application's responses:

- **Retrieval Augmented Generation (RAG)**: A technique that involves using a data source to provide grounding context to prompts. RAG can be a useful approach when you need the model to **answer questions based on a specific knowledge domain** or **when you need the model to consider information related to events that occurred after the training data on which the model is based**.
- **Fine-tuning**: A technique that involves extending the training of a foundation model by **providing example prompts and responses that reflect the desired output format and style**.

Both of these approaches involve additional cost, complexity, and maintainability challenges, so as a general rule **it's best to start your optimization efforts through prompt engineering, and then consider additional strategies if necessary**.

The strategy you should choose as a developer depends on your requirements:

image



- **Optimize for context**: When the model ***lacks contextual knowledge*** and you want to maximize responses accuracy.
- **Optimize the model**: When you want to **improve the response format, style, or speech** by maximizing consistency of behavior.

To optimize for context, you can apply a **Retrieval Augmented Generation (RAG)** pattern. With RAG, you ground your data by first retrieving context from a data source before generating a response. For example, you want employees to ask questions about expense claim processes and limits based on your own corporation's expenses policy documentation.

When you want the model to respond in a specific style or format, you can instruct the model to do so by adding guidelines in the system message. When you notice the model's behavior isn't consistent, you can further enforce consistency in behavior by fine-tuning a model. With **fine-tuning**, you **train a base language model on a dataset of example prompts and responses** before integrating it in your application, with the result that the fine-tuned model will produce responses that are consistent with the examples in the fine-tuning training dataset.

You can use any combination of optimization strategies, for example **prompt engineering, RAG, and a fine-tuned model**, to improve your language application.

# Exercise - Explore, deploy, and chat with language models

## Choose and deploy a language model

The Azure AI Foundry model catalog serves as a central repository where you can explore and use a variety of models, facilitating the creation of your generative AI scenario.

In this exercise, you'll explore the model catalog in Azure AI Foundry portal, and compare potential models for a generative AI application that assists in solving problems.

# Module assessment

1. Where can you test a deployed model in the Azure AI Foundry portal? **Chat playground**

2. You want to **specify the tone, format, and content** for each interaction with your model in the playground. What should you use to customize the model response? **System message**
3. What deployment option should you choose to host an OpenAI model in an Azure AI Foundry resource? Standard deployment

# Summary

In this module, you learned how to:

- Select a language model from the model catalog.
- Deploy a model to an endpoint.
- Test a model and improve the performance of the model.

# Develop an AI app with the Azure AI Foundry SDK

Use the Azure AI Foundry SDK to develop AI applications with Azure AI Foundry projects.

## Learning objectives

After completing this module, you'll be able to:

- Describe **capabilities of the Azure AI Foundry SDK**.
- Use the Azure AI Foundry SDK to work with connections in projects.
- Use the Azure AI Foundry SDK to **develop an AI chat app**.

## Introduction

Developers creating AI solutions with Azure AI Foundry need to work with a combination of services and software frameworks. The **Azure AI Foundry SDK** is designed to bring together common services and code libraries in an AI project through a central programmatic access point, making it easier for developers to write the code needed to build effective AI apps on Azure.

In this module, you'll learn how to use the Azure AI Foundry SDK to work with resources in an AI project.

> Note: Azure AI Foundry SDK is currently in public preview. Details described in this module are subject to change.

## What is the Azure AI Foundry SDK?

Azure AI Foundry provides a **REST API** that you can use to work with AI Foundry projects and the resources they contain. Additionally, **multiple language-specific SDKs** are available, enabling developers to write code that uses resources in an Azure AI Foundry project in their preferred development language. With an 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 core package for working with projects is the **Azure AI Projects library**, which enables you to connect to an Azure AI Foundry project and access the resources defined within it. Available language-specific packages the for Azure AI Projects library include:

- Azure AI Projects for Python
- Azure AI Projects for Microsoft .NET
- Azure AI Projects for JavaScript

> Note: In this module, we'll use **Python** code examples for common tasks that a developer may need to perform with Azure AI Foundry projects. You can refer to the other language-specific SDK documentation to find equivalent code for your preferred language. Each SDK is developed and maintained independently, so some functionality may be at different stages of implementation for each language.

To use the Azure AI Projects library in Python, you can use the pip package installation utility to install the azure-ai-projects package from PyPi:

```
pip install azure-ai-projects
```

## 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.

> Note: 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, as shown in this Python example:

```
from azure.identity import DefaultAzureCredential
from azure.ai.projects import AIProjectClient
...

project_endpoint = "https://....."
project_client = AIProjectClient(
    credential=DefaultAzureCredential(),
    endpoint=project_endpoint)
```

> Note: The code uses the default Azure credentials to authenticate when accessing the project. To enable this authentication, in addition to the **azure-ai-projects** package, you need to install the azure-identity package:
> `pip install azure-identity`

> Tip: To access the project successfully, the code must be run in the context of an authenticated Azure session. For example, you could use the Azure command-line interface (CLI) `az-login` command to sign in before running the code.

# 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.

For example, 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.

The connection objects returned by these methods include connection-specific properties, including credentials, which you can use to connect to the associated resource.

The following code example lists all of the resource connections that have been added to a project:

```
from azure.identity import DefaultAzureCredential
from azure.ai.projects import AIProjectClient

try:

    # Get project client
    project_endpoint = "https://....."
    project_client = AIProjectClient(
            credential=DefaultAzureCredential(),
            endpoint=project_endpoint,
        )

    ## List all connections in the project
    connections = project_client.connections
    print("List all connections:")
    for connection in connections.list():
        print(f"{connection.name} ({connection.type})")

except Exception as ex:
    print(ex)
```

# Create a chat client

A common scenario in an AI application is to **connect to a generative AI model and use prompts to engage in a chat-based dialog** with it.

While you can use the Azure OpenAI SDK, to connect "directly" to a model using key-based or Microsoft Entra ID authentication; when your model is deployed in an Azure AI Foundry project, you can also use the Azure AI Foundry SDK to retrieve a project client, from which you can then get an authenticated OpenAI chat client for any models deployed in the project's Azure AI Foundry resource. This approach makes it easy to write code that consumes models deployed in your project, switching between them easily by changing the model deployment name parameter.

> Tip: You can use the OpenAI chat client provided by an Azure AI Foundry project to chat with any model deployed in the associated Azure AI Foundry resource - even non-OpenAI models, such as Microsoft Phi models.

The following Python code sample uses the `get_openai_client()` method to get an OpenAI client with which to chat with a model that has been deployed in the project'a Azure AI Foundry resource.

```
from azure.identity import DefaultAzureCredential
from azure.ai.projects import AIProjectClient
from openai import AzureOpenAI

try:

    # connect to the project
    project_endpoint = "https://....."
    project_client = AIProjectClient(
            credential=DefaultAzureCredential(),
            endpoint=project_endpoint,
        )

    # Get a chat client
    chat_client = project_client.get_openai_client(api_version="2024-10-21")

    # Get a chat completion based on a user-provided prompt
    user_prompt = input("Enter a question:")

    response = chat_client.chat.completions.create(
        model=your_model_deployment_name,
        messages=[
            {"role": "system", "content": "You are a helpful AI assistant."},
            {"role": "user", "content": user_prompt}
        ]
    )
    print(response.choices[0].message.content)

except Exception as ex:
    print(ex)
```

> Note: In addition to the `azure-ai-projects` and `azure-identity` packages discussed previously, the sample code shown here assumes that the openai package has been installed:
>
> `pip install openai`

# Exercise - Create a generative AI chat app

## Create a generative AI chat app

In this exercise, you use the Azure AI Foundry Python SDK to create a simple chat app that connects to a project and chats with a language model.

# Module assessment

1. i. What class in the Azure AI Foundry SDK provides a proxy object for a project? **AIProjectClient**
2. What value is needed to instantiate a `AIProjectClient` object? **The project endpoint**.
3. Which SDK should you use to chat with a model that is deployed in an Azure AI Foundry resource? **Azure OpenAI**

# Summary

By using the **Azure AI Foundry SDK**, you can develop rich AI applications that **use resources in your Azure AI Foundry projects**. The Azure AI Foundry SDK `AIProjectClient` class provides a programmatic proxy for a project, enabling you to access connected resources and to use service-specific libraries to consume them.

# Get started with prompt flow to develop language model apps in the Azure AI Foundry

Learn about how to use prompt flow to develop applications that leverage language models in the Azure AI Foundry.

## Learning objectives

By the end of this module, you'll be able to:

- Understand the development lifecycle when creating language model applications.
- Understand what a flow is in **prompt flow**.
- Explore the core components when working with prompt flow.

## Introduction

The true power of **Large Language Models (LLMs)** lies in their application. Whether you want to use LLMs to classify web pages into categories, or to build a chatbot on your data. To harness the power of the LLMs available, you need to **create an application that combines your data sources with LLMs and generates the desired output**.

To develop, test, tune, and deploy LLM applications, you can use **prompt flow**, accessible in the Azure Machine Learning studio and the Azure AI Foundry portal.

> Note: The focus of this module is on **understanding and exploring prompt flow through Azure AI Foundry**. However, note that the content applies to the prompt flow experience in both Azure Machine Learning and Azure AI Foundry.

**Prompt flow takes a prompt as input**, which in the context of LLMs, refers to the query provided to the LLM application to generate a response. It's the text or set of instructions given to the LLM application, prompting it to generate output or perform a specific task.

For example, when you want to use a text generation model, the prompt might be a sentence or a paragraph that initiates the generation process. In the context of a question-answering model, the prompt could be a query asking for information on a particular topic. The effectiveness of the prompt often depends on how well it conveys the user's intent and the desired outcome.

**Prompt flow allows you to create flows, which refers to the sequence of actions or steps that are taken to achieve a specific task or functionality**. **A flow represents the overall process or pipeline that incorporates the interaction with the LLM to address a particular use case**. The flow encapsulates the entire journey from receiving input to generating output or performing a desired action.

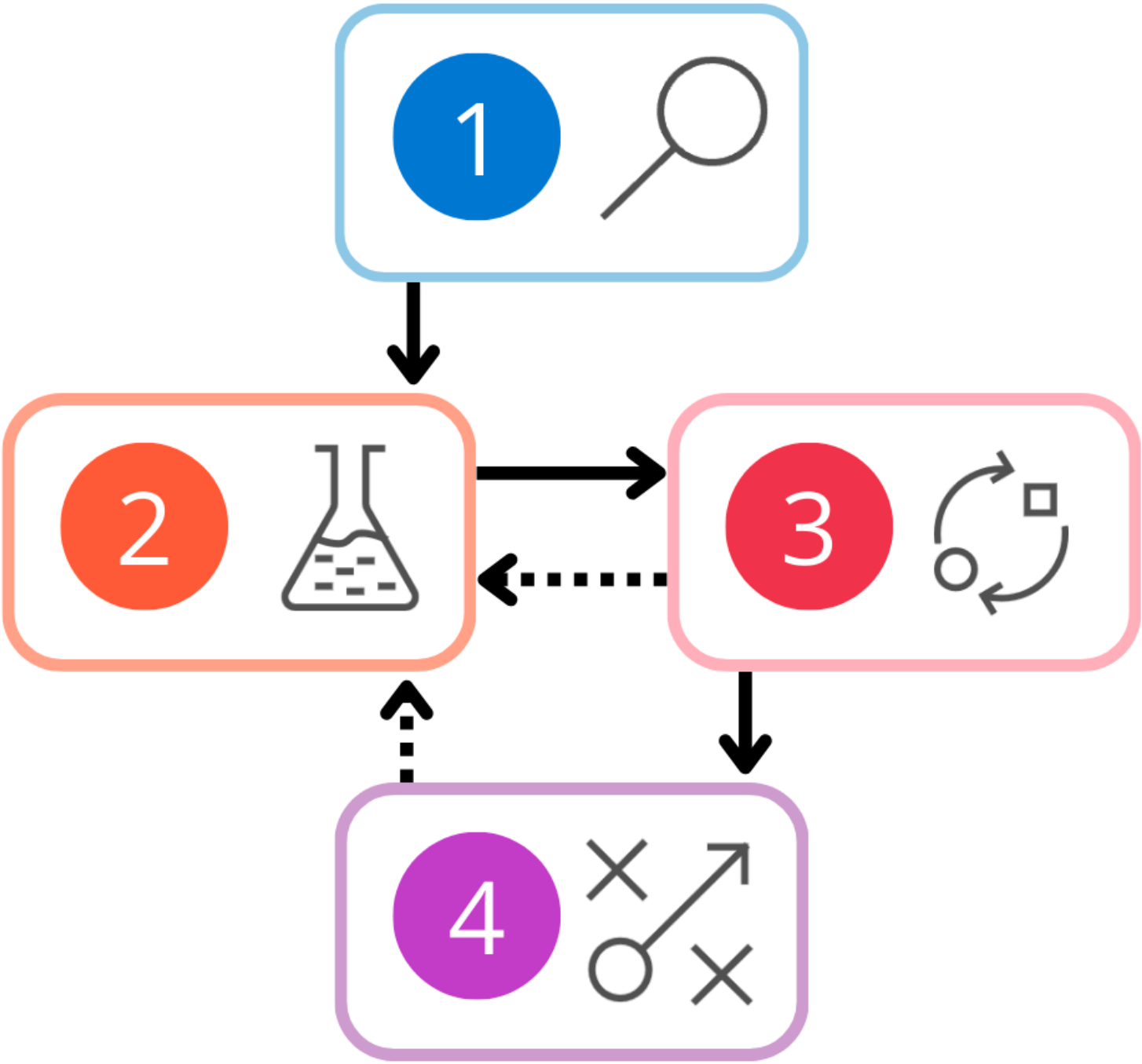
## Understand the development lifecycle of a large language model (LLM) app

Before understanding how to work with prompt flow, let's explore the **development lifecycle of a Large Language Model (LLM) application**.

The lifecycle consists of the following stages:

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

During both **3) evaluation and refinement, and 4) production**, you might find that your solution needs to be improved. You can **revert back to 2) experimentation** during which you develop your flow continuously, until you're satisfied with the results.




Let's explore each of these phases in more detail.

## Initialization

Imagine you want to design and develop an LLM application to classify news articles. Before you start creating anything, you need to define what categories you want as output. You need to understand **what a typical news article looks like, how you present the article as input to your application, and how the application generates the desired output**.

In other words, during initialization you:

1. **Define** the objective

2. **Collect** a sample dataset
3. **Build** a basic prompt
4. **Design** the flow

To design, develop, and test an LLM application, you need a sample dataset that serves as the input. **A sample dataset is a small representative subset of the data you eventually expect to parse as input to your LLM application**.

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 of news articles, and decided on which categories you want the articles to be classified into. You designed a flow that takes a news article as input, and uses an LLM to classify the article. 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.
2. **Evaluate** the prompt's performance.
3. If you're satisfied with the result, you can **move on to evaluation and refinement**.
4. If you think there's room for improvement, you can **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 news articles, 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

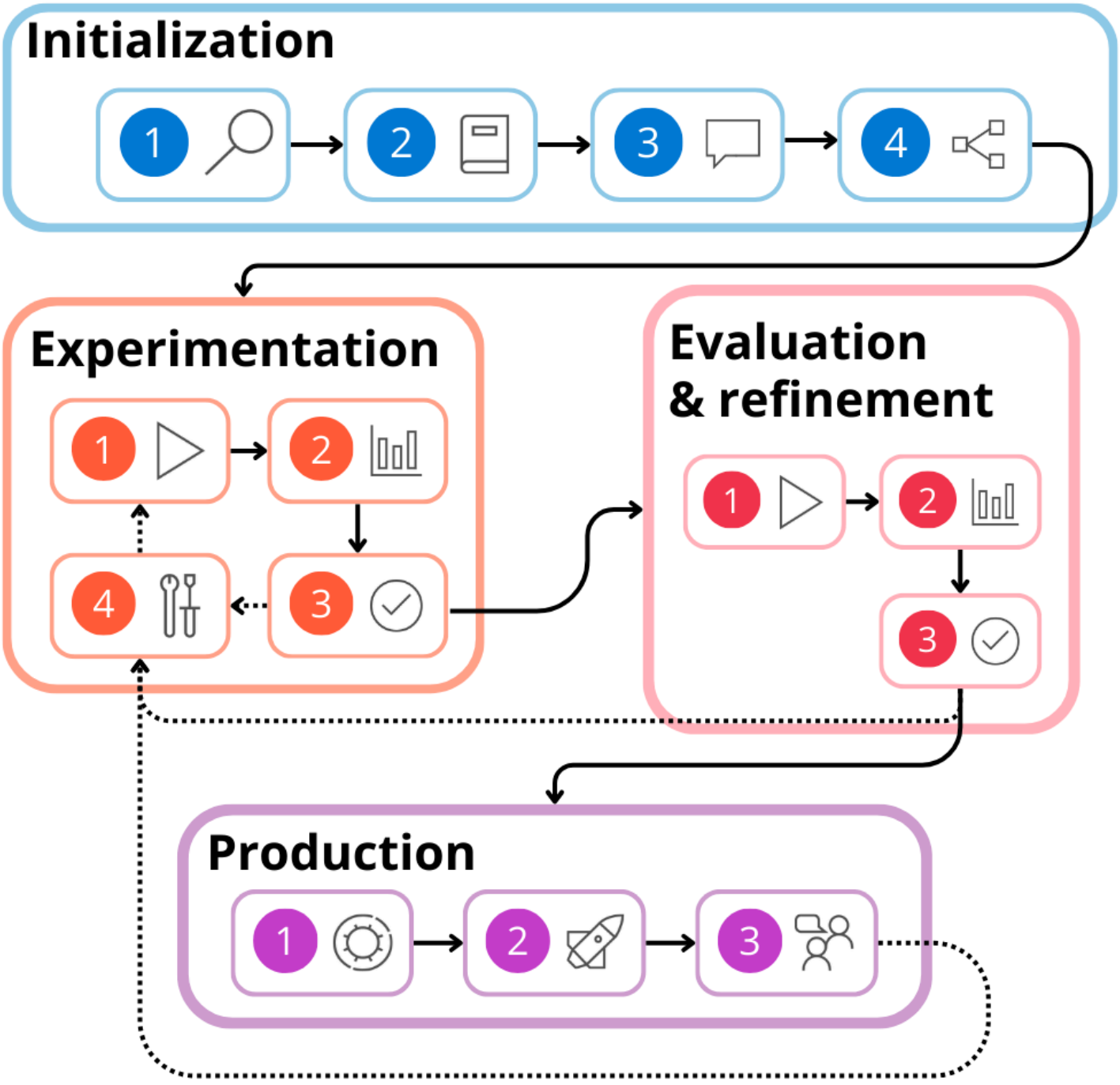
Finally, your news article classification application is ready for production.

During production, you:

1. **Optimize** the flow that classifies incoming articles for efficiency and effectiveness.
2. **Deploy** your flow to an **endpoint**. When you call the endpoint, the flow is triggered to run and the desired output is generated.
3. **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.

## Explore the complete development lifecycle

Now that you understand each stage of the development lifecycle of an LLM application, you can explore the complete overview.




.

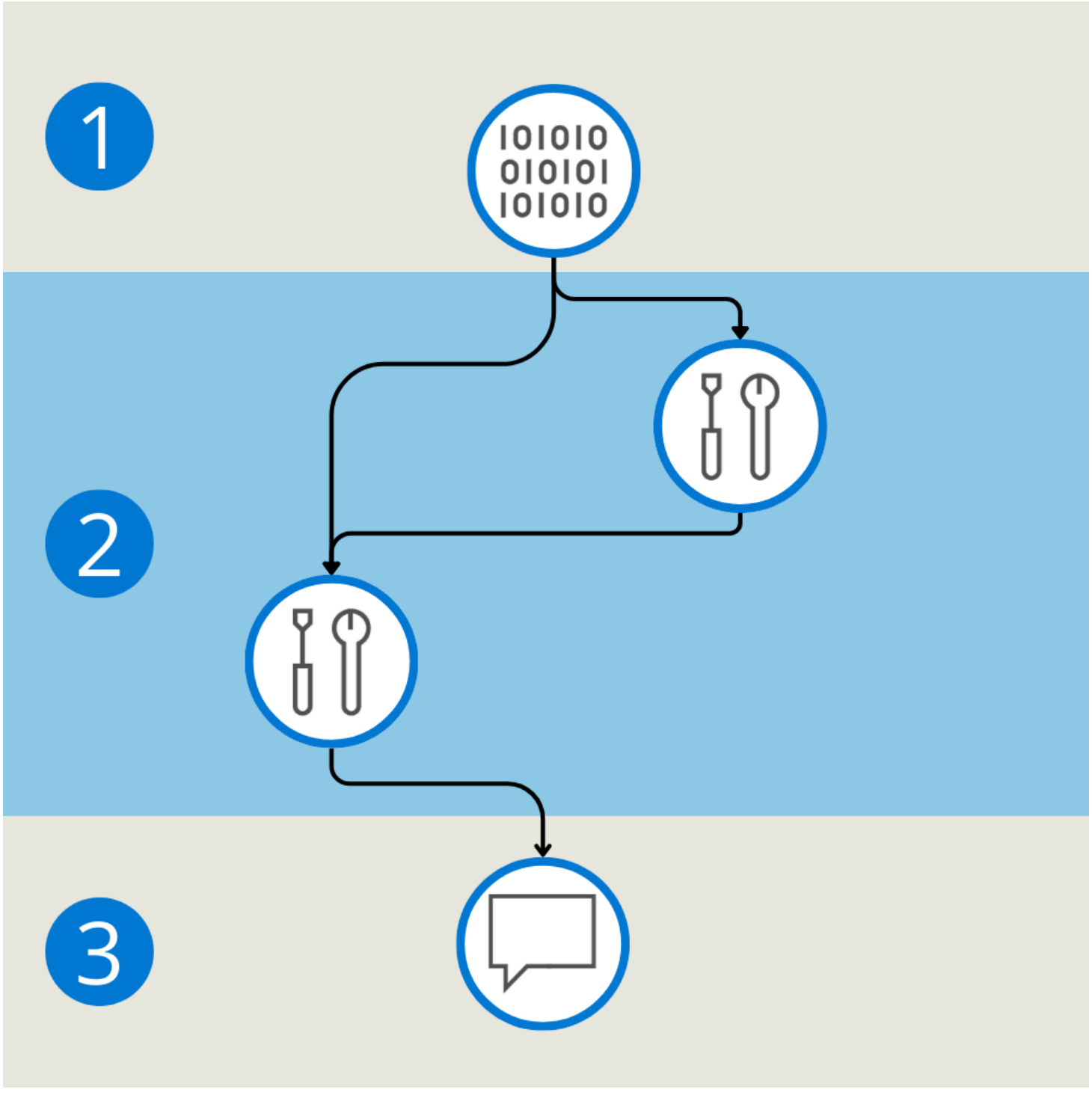
## Understand core components and explore flow types

To create **a Large Language Model (LLM) application with prompt flow**, you need to understand prompt flow's core components.

## 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:

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



Similar to a pipeline, **a flow can consist of multiple nodes that can use the flow's inputs or any output generated by another node**. You can add a node to a flow by choosing one of the available types of tools.

## 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.

> Tip: If you're looking for functionality that is not offered by the available tools, you can create your own custom tool.

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.

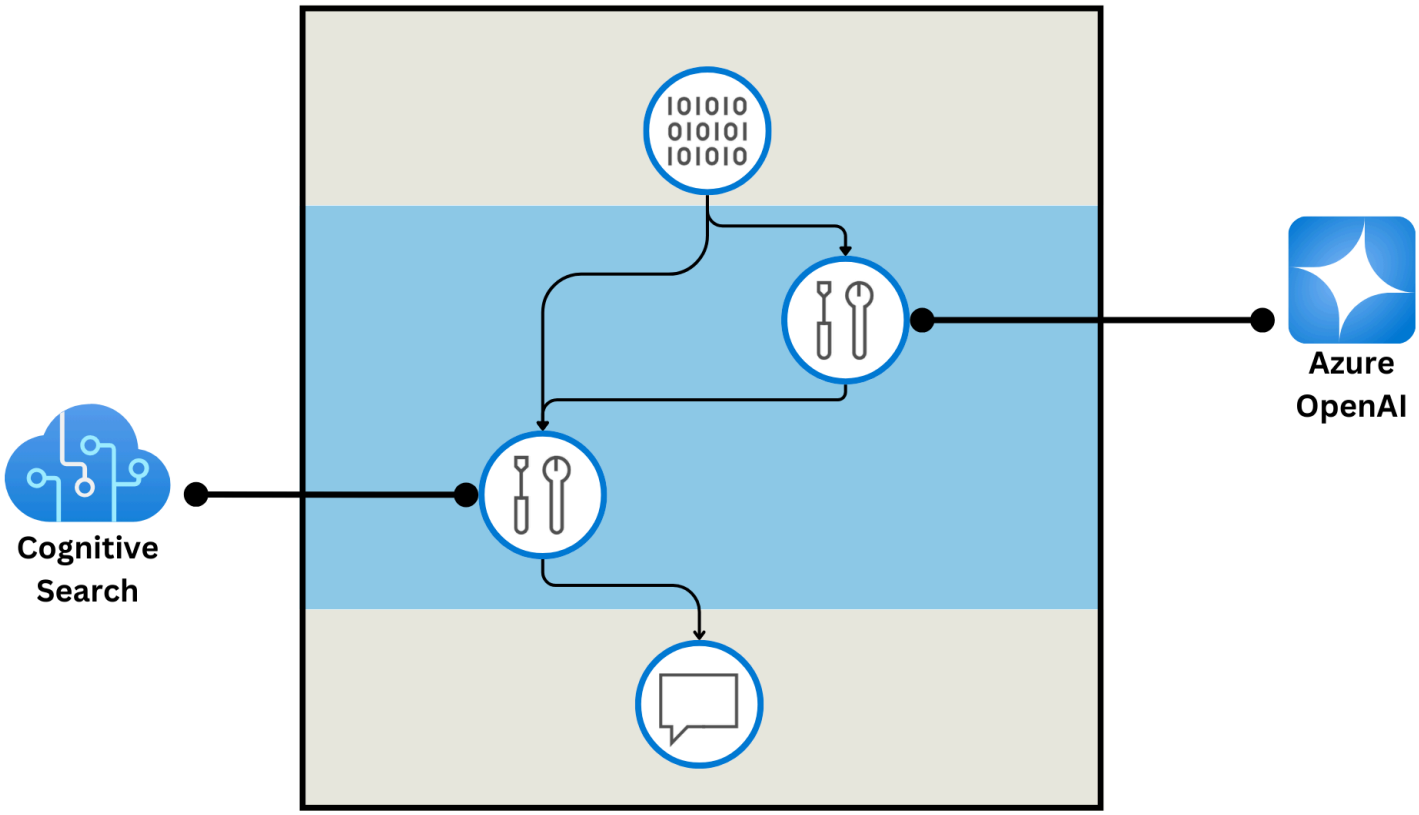
Now that you understand how a flow is structured and what you can use it for, let's explore how you can create a flow.

# 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.

Certain built-in tools require you to have a connection configured:

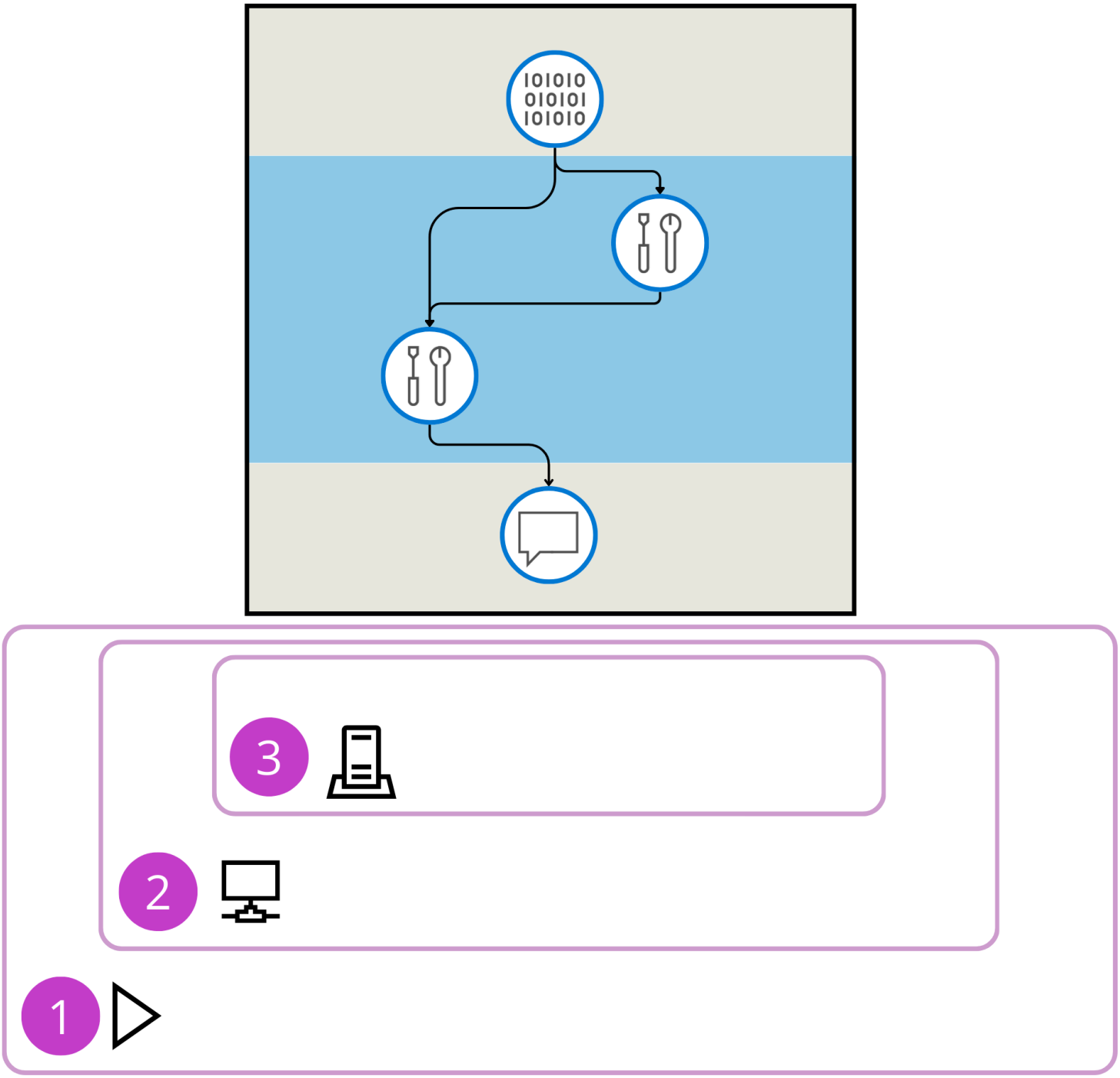
| Connection type | Built-in tools |
| --- | --- |
| Azure OpenAI | LLM or Python |

| Connection type | Built-in tools |
| --- | --- |
| OpenAI | LLM or Python |
| Azure AI Search | Vector DB Lookup or Python |
| Serp | Serp API or Python |
| Custom | Python |

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 (1) are a combination of a **compute instance** (2) providing the necessary **compute resources**, and an environment (3) 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.

# 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:

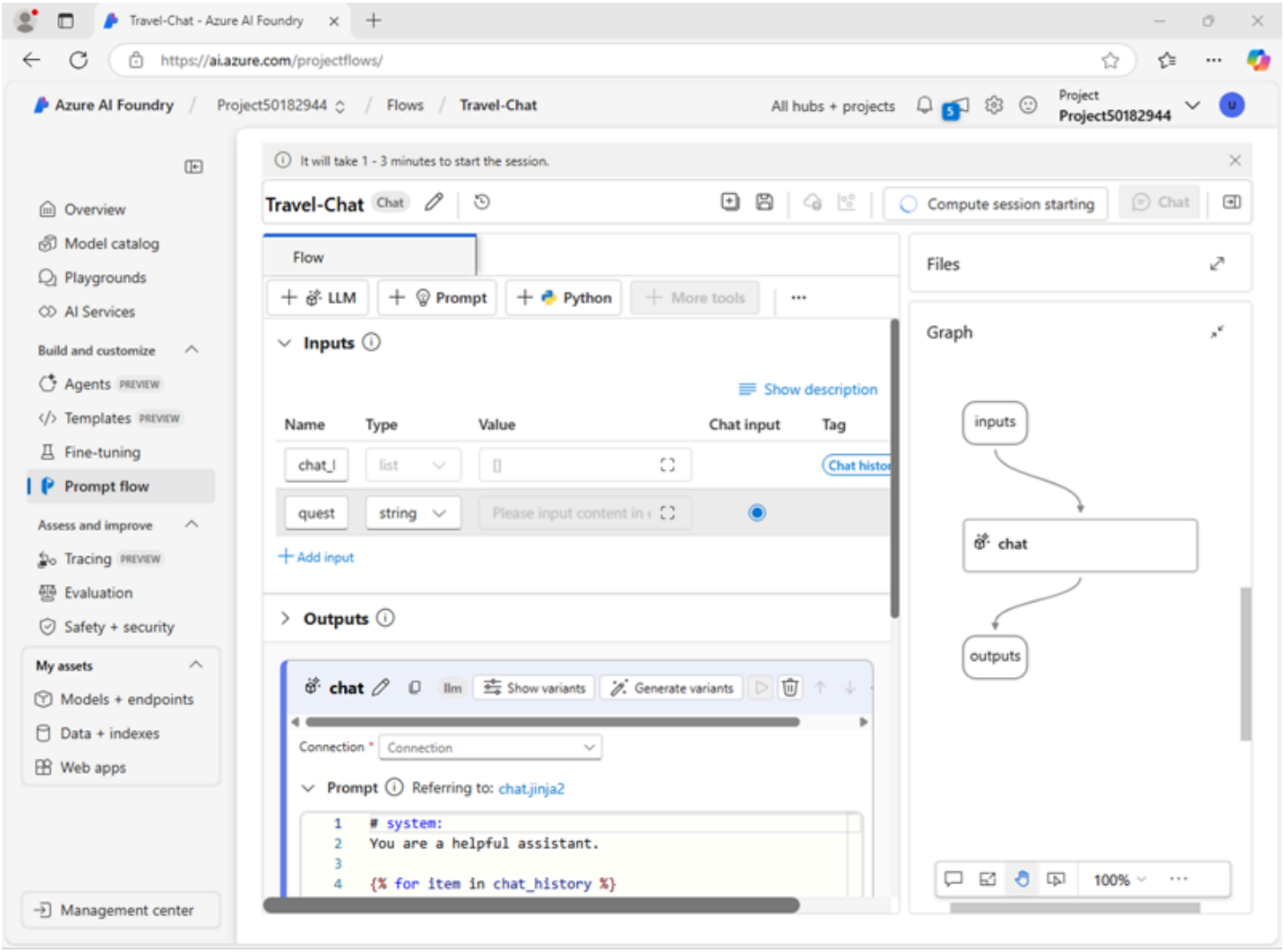
- **Groundedness**: Measures alignment of the LLM application's output with the input source or database.
- **Relevance**: Assesses how pertinent the LLM application's output is to the given input.
- **Coherence**: Evaluates the logical flow and readability of the LLM application's text.
- **Fluency**: 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.

# Exercise - Get started with prompt flow

## Use a prompt flow to manage conversation in a chat app

In this exercise, you'll use Azure AI Foundry portal's prompt flow to create a custom chat app that uses a user prompt and chat history as inputs, and uses a GPT model from Azure OpenAI to generate an output.




# Module assessment

1. A flow uses an LLM tool to generate text with a GPT-3.5 model. What do you need to create to ensure prompt flow can securely call the deployed model from Azure OpenAI? **Connections**.
2. You want to integrate your flow with an online website. What do you need to do to easily integrate your flow? **Deploy your flow to an endpoint**.
3. After deployment, you notice that your flow is underperforming. Which stage in the development lifecycle should you revert back to? **Experimentation**.

# Summary

In this module, you learned:

- The development lifecycle when creating LLM applications.
- What a flow is in **prompt flow**.
- The core components when working with prompt flow.

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

**Retrieval Augmented Generation (RAG)** is a common pattern used in generative AI solutions to *ground* prompts with your data. Azure AI Foundry provides support for **adding data, creating indexes, and integrating them with generative AI models** to help you build RAG-based solutions.

## Learning objectives

By the end of this module, you'll be able to:

- Identify the need to **ground your language model with Retrieval Augmented Generation (RAG)**.
- **Index your data with Azure AI Search** to make it searchable for language models
- Build an agent using RAG on your own data in the Azure AI Foundry portal

## Introduction

Language models are growing in popularity as they create impressive coherent answers to a user's questions. Especially when a user interacts with a language model through chat, it provides an intuitive way to get the information they need.

One prevalent challenge when implementing language models through chat is the so-called **groundedness**, which refers to **whether a response is rooted, connected, or anchored in reality or a specific context**. In other words, **groundedness refers to whether the response of a language model is based on factual information**.

### 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. For example, the question "Which product should I use to do XYZ?" might include details of a fictional product.

## 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.

In this module, you explore how to **create your own chat-based language model application that is grounded, by building an agent with your own data**.

# 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 agent 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**.

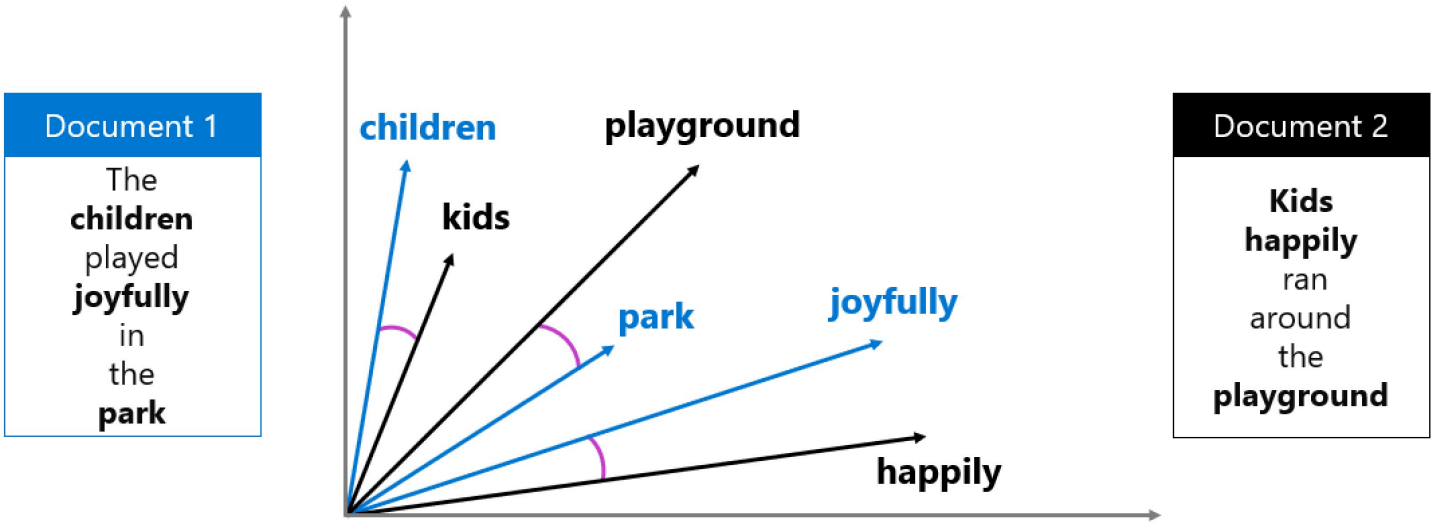
An **embedding** is a special format of data representation that a search engine can use to easily find the relevant information. More specifically, **an embedding is a vector of floating-point numbers**.

For example, imagine you have two documents with the following contents:

- "The children played joyfully in the park."
- "Kids happily ran around the playground."

These two documents contain texts that are semantically related, even though different words are used. By creating vector embeddings for the text in the documents, the relation between the words in the text can be mathematically calculated.

Imagine the keywords being extracted from the document and plotted as a vector in a multidimensional space:




The distance between vectors can be calculated by measuring the cosine of the angle between two vectors, also known as the **cosine similarity**. In other words, the cosine similarity computes the **semantic similarity** between documents and a query.

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.

> Tip: Learn more about embeddings in the Azure OpenAI Service.

## 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.

> Tip: Learn more about vector search.

## Searching an index

There are several ways that information can be queried in 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).

## Create a RAG-based client application

When you've created an Azure AI Search index for your contextual data, you can use it with an OpenAI model. To ground prompts with data from your index, the Azure OpenAI SDK supports extending the request with connection details for the index.

The following Python code example shows how to implement this pattern.

```
from openai import AzureOpenAI

# Get an Azure OpenAI chat client {#get-an-azure-openai-chat-client }
chat_client = AzureOpenAI(
    api_version = "2024-12-01-preview",
    azure_endpoint = open_ai_endpoint,
    api_key = open_ai_key
)

# Initialize prompt with system message {#initialize-prompt-with-system-message }
prompt = [
    {"role": "system", "content": "You are a helpful AI assistant."}
]

# Add a user input message to the prompt {#add-a-user-input-message-to-the-prompt }
input_text = input("Enter a question: ")
prompt.append({"role": "user", "content": input_text})

# Additional parameters to apply RAG pattern using the AI Search index {#additional-parameters-t
rag_params = {
    "data_sources": [
        {
            "type": "azure_search",
            "parameters": {
                "endpoint": search_url,
                "index_name": "index_name",
                "authentication": {
                    "type": "api_key",
                    "key": search_key,
                }
            }
        }
    ],
}

# Submit the prompt with the index information {#submit-the-prompt-with-the-index-information }
response = chat_client.chat.completions.create(
    model="<model_deployment_name>",
    messages=prompt,
    extra_body=rag_params
)

# Print the contextualized response {#print-the-contextualized-response }
```

```
completion = response.choices[0].message.content
print(completion)
```

In this example, the search against the index is **keyword-based** - in other words, the query consists of the text in the user prompt, which is matched to text in the indexed documents. When using an index that supports it, an alternative approach is to use a **vector-based query** in which the index and the query use numeric vectors to represent text tokens. Searching with vectors enables matching based on semantic similarity as well as literal text matches.

To use a vector-based query, you can modify the specification of the Azure AI Search data source details to include an embedding model; which is then used to vectorize the query text.

```
rag_params = {
    "data_sources": [
        {
            "type": "azure_search",
            "parameters": {
                "endpoint": search_url,
                "index_name": "index_name",
                "authentication": {
                    "type": "api_key",
                    "key": search_key,
                },
                # Params for vector-based query
                "query_type": "vector",
                "embedding_dependency": {
                    "type": "deployment_name",
                    "deployment_name": "<embedding_model_deployment_name>",
                },
            }
        }
    ],
}
```

# Implement RAG in a prompt flow

After uploading data to Azure AI Foundry and creating an index on your data using the integration with Azure AI Search, you can implement the **RAG pattern with *Prompt Flow*** to build a generative AI

application.

**Prompt Flow is a development framework for defining flows that orchestrate interactions with an LLM**.

A flow begins with one or more inputs, usually a question or prompt entered by a user, and in the case of iterative conversations the chat history to this point.

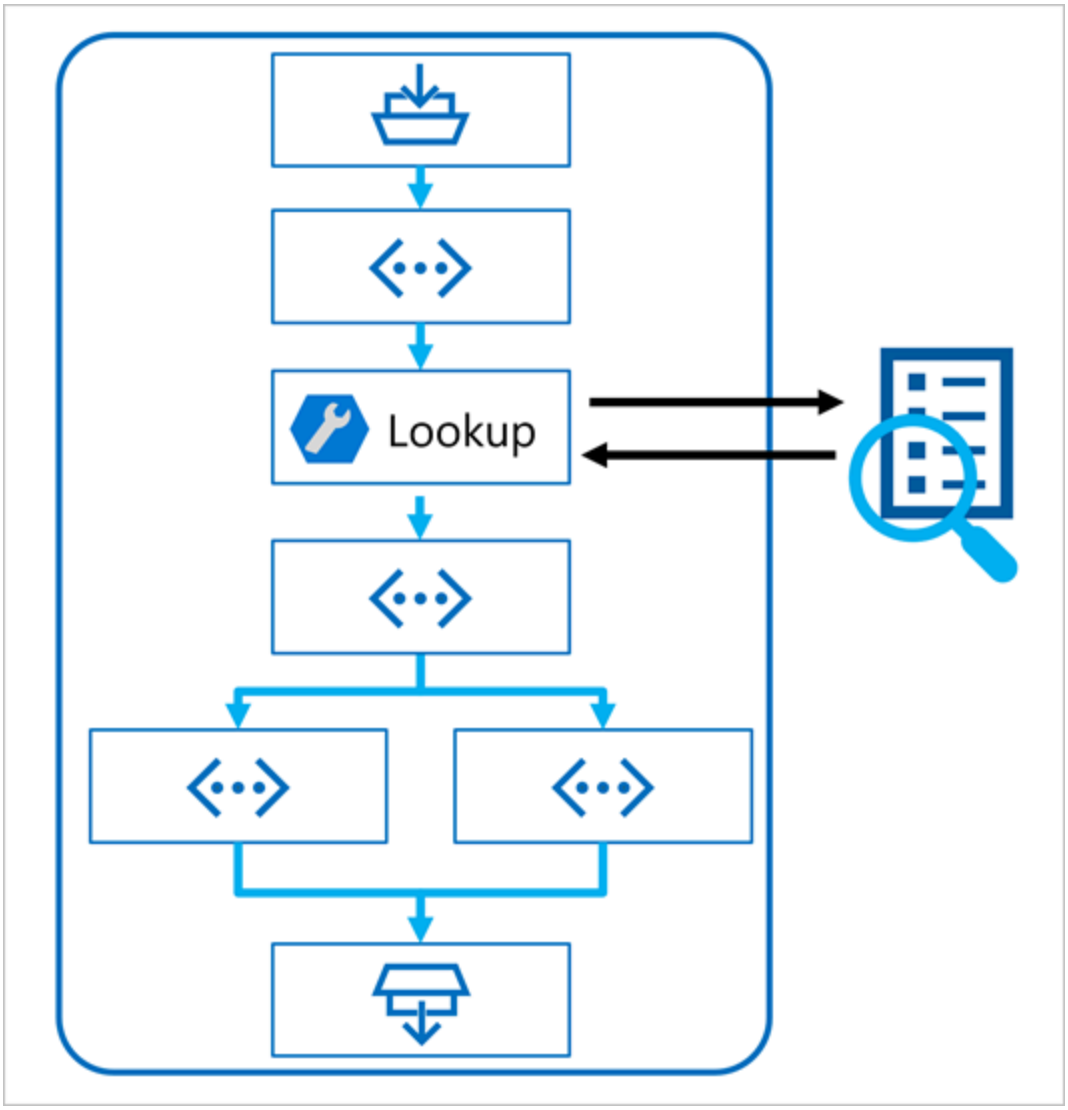
The flow is then defined as a series of connected tools, each of which performs a specific operation on the inputs and other environmental variables. There are multiple types of tool that you can include in a prompt flow to perform tasks such as:

- Running custom Python code
- Looking up data values in an index
- Creating prompt variants - enabling you to define multiple versions of a prompt for a large language model (LLM), varying system messages or prompt wording, and compare and evaluate the results from each variant.
- Submitting a prompt to an LLM to generate results.

Finally, the flow has one or more outputs, typically to return the generated results from an LLM.

## Using the RAG pattern in a prompt flow

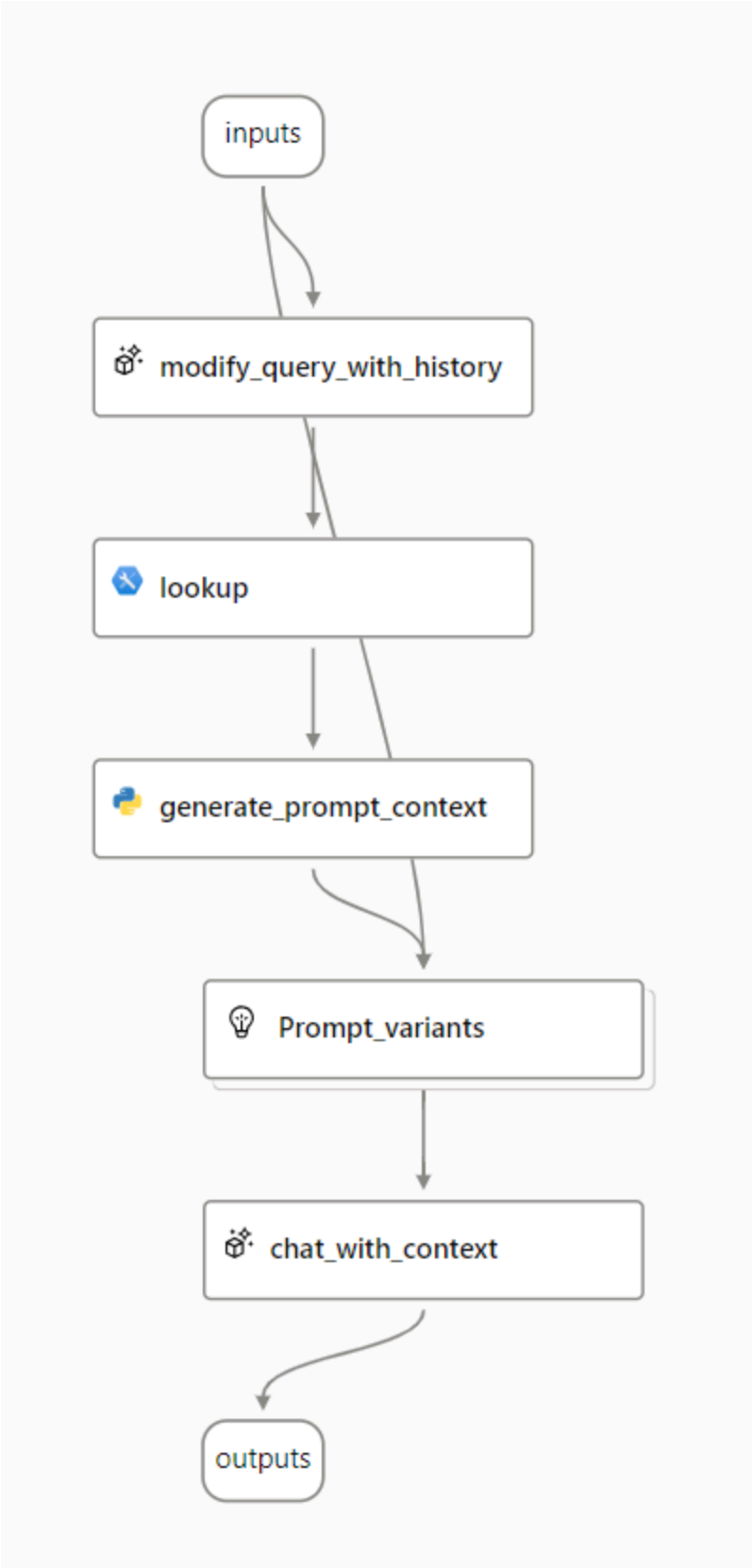
The key to using the RAG pattern in a prompt flow is to use **an Index Lookup tool to retrieve data from an index so that subsequent tools in the flow can use the results to augment the prompt used to generate output from an LLM**.




## Use a sample to create a chat flow

Prompt flow provides various samples you can use as a starting point to create an application. When you want to combine RAG and a language model in your application, you can clone the Multi-round Q&A on your data sample.

The sample contains the necessary elements to include RAG and a language model:




1. Append the history to the chat input to define a prompt in the form of a contextualized form of a question.
2. Look up relevant information from your data using your search index.
3. Generate the prompt context by using the retrieved data from the index to augment the question.
4. Create prompt variants by adding a system message and structuring the chat history.

5. Submit the prompt to a language model that generates a natural language response.

Let's explore each of these elements in more detail.

## 1. Modify query with history

The first step in the flow is a Large Language Model (LLM) node that **takes the chat history and the user's last question and generates a new question that includes all necessary information**. By doing so, you generate more succinct input that is processed by the rest of the flow.

## 2. Look up relevant information

Next, you use the **Index Lookup tool** to query the search index you created with the integrated Azure AI Search feature and find the relevant information from your data source.

> Tip: Learn more about the Index Lookup tool.

## 3. Generate prompt context

The output of the **Index Lookup tool** is the retrieved context you want to use when generating a response to the user. You want to use the output in a prompt that is sent to a language model, which means you want to parse the output into a more suitable format.

**The output of the Index Lookup tool can include the top n results** (depending on the parameters you set). When you generate the prompt context, you can use a Python node to iterate over the retrieved documents from your data source and combine their contents and sources into one document string. The string will be used in the prompt you send to the language model in the next step of the flow.

## 4. Define prompt variants

When you construct the prompt you want to send to your language model, you can **use variants to represent different prompt contents**.

When including RAG in your chat flow, your goal is to ground the chatbot's responses. Next to retrieving relevant context from your data source, you can also influence the groundedness of the chatbot's response by instructing it to use the context and aim to be factual.

With the prompt variants, you can provide varying system messages in the prompt to explore which content provides the most groundedness.

## 5. Chat with context

Finally, you use an LLM node to send the prompt to a language model to generate a response using the relevant context retrieved from your data source. The response from this node is also the output of the entire flow.

After configuring the sample chat flow to use your indexed data and the language model of your choosing, you can deploy the flow and integrate it with an application to offer users an agentic experience.

# Exercise - Create a generative AI app that uses your own data

## Create a generative AI app that uses your own data

**Retrieval Augmented Generation (RAG)** is a technique used to build applications that integrate data from custom data sources into a prompt for a generative AI model. RAG is a commonly used pattern for developing generative AI apps - chat-based applications that use a language model to interpret inputs and generate appropriate responses.

In this exercise, you'll use **Azure AI Foundry** to integrate custom data into a generative AI solution.

# Module assessment

1. What does groundedness refer to in the context of generative AI? **Using data to contextualize prompts and ensure relevant responses**.
2. What pattern can you use to ground prompts? Retrieval Augmented Generation (RAG)
3. How can you use the RAG pattern in a client app that uses the Azure OpenAI SDK? **Add index connection details to the OpenAI ChatClient configuration**.

# Summary

In this module, you learned to:

- Identify the need to ground your language model with Retrieval Augmented Generation (RAG).
- Index your data with Azure AI Search to make it searchable for language models.
- Build an agent using RAG on your own data in Azure AI Foundry.

# Fine-tune a language model with Azure AI Foundry

Train a base language model on a chat-completion task. The model catalog in Azure AI Foundry offers many open-source models that can be **fine-tuned for your specific model behavior needs**.

## Learning objectives

By the end of this module, you'll be able to:

- Understand **when to fine-tune a model**.
- Prepare your data to fine-tune a chat completion model.
- Fine-tune a base model in the Azure AI Foundry portal.

## Introduction

Language models are pretrained models that provide you with a great starting point. By using one of the available base or foundation models, you can save time and effort as you need less data to train a model for your specific use case.

Imagine you're a developer working for a travel agency. When customers use your chat application to get help with their travel-related questions, **you want the responses to be in a specific format and style**. Your company has **a specific tone of voice** that resonates with your client base. The marketing department finds it important that the chat application is aligned with your company's tone of voice too.

There are various strategies to optimize the model's behavior and the performance of your chat application. One strategy is to **fine-tune a language model**, which you can then integrate with your chat application. **The benefit of fine-tuning over training your own language model, is that you need less time, compute resources, and data to customize the model to your needs**.

In this module, you learn how to fine-tune a base model from the model catalog in the Azure AI Foundry portal, that you can then integrate in a chat application.

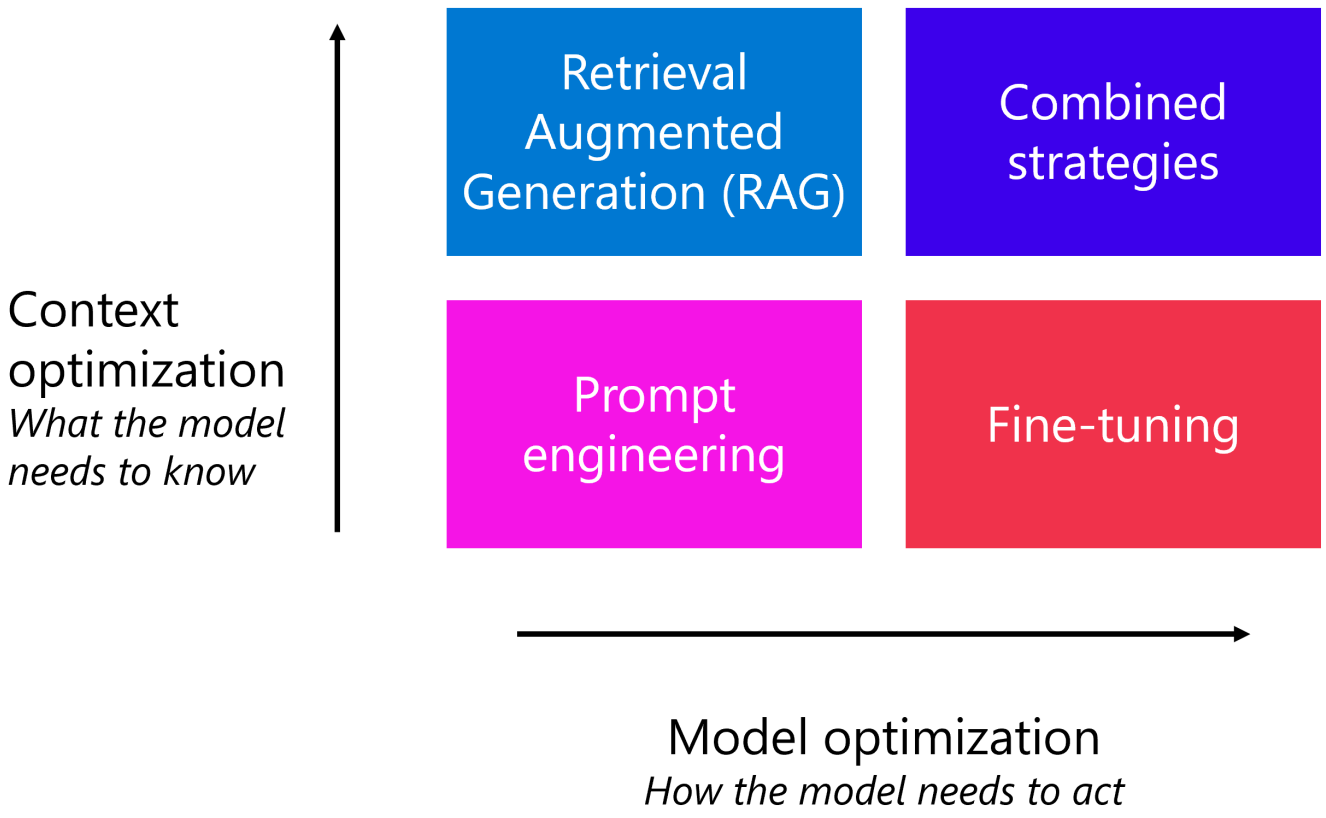
# Understand when to fine-tune a language model

Before you start fine-tuning a model, you need to have a clear understanding of **what fine-tuning is and when you should use it**.

When you want to develop a chat application with Azure AI Foundry, you can use **prompt flow** to create a chat application that is integrated with a language model to generate responses. To improve the quality of the responses the model generates, you can try various strategies. **The easiest strategy is to apply prompt engineering**. You can **change the way you format your question**, but you can also **update the system message** that is sent along with the prompt to the 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**. For example, you want customers to ask questions about hotels that

you're offering in your travel booking catalog. 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.

How the model needs to act mostly relates to the style, format, and tone of the responses generated by a model. When you want your model to **adhere to a specific style and format** when responding, you can instruct the model to do so through prompt engineering too. Sometimes however, prompt engineering might not lead to consistent results. It can still happen that a model ignores your instructions and behaves differently.

Within prompt engineering, a technique used to "force" the model to generate output in a specific format, is to **provide the model with various examples of what the desired output might look like**, also known as **one-shot (one example) or few-shot (few examples)**. Still, it can happen that your model doesn't always generate the output in the style and format you specified.

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.

When you decide you want to fine-tune a language model, you need to **identify the dataset you can use to fine-tune your language model**.

Similar to any machine learning model, **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

The three variables come together in **a JSON Lines or JSONL file**. For example, one line in such a dataset might look like:

```
{"messages": [{"role": "system", "content": "You are an Xbox customer support agent whose primar
```

```
{
    "messages": [
        {
            "role": "system",
            "content": "You are an Xbox customer support agent whose primary goal is to help use
        },
        {
            "role": "user",
            "content": "Is Xbox better than PlayStation?"
        },
        {
            "role": "assistant",
            "content": "I apologize, but I cannot provide personal opinions. My primary job is t
        }
    ]
}
```

The dataset should show the model's ideal behavior. You can create this dataset based on the chat history of a chat application you have. 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**.

You can include multiple turns of a conversation on a single line in the dataset. If you want to fine-tune only on specific assistant messages, you can optionally use the `weight` key-value pair. **When the weight is set to 0, the message is ignored, when you set to 1, the message is included for training**.

An example of a multi-turn chat file format with weights:

```
{
    "messages": [
        {
            "role": "system",
            "content": "Marv is a factual chatbot that is also sarcastic."
        },
        {
            "role": "user",
            "content": "What's the capital of France?"
        },
        {
            "role": "assistant",
            "content": "Paris",
            "weight": 0
        },
        {
            "role": "user",
            "content": "Can you be more sarcastic?"
        },
        {
            "role": "assistant",
            "content": "Paris, as if everyone doesn't know that already.",
            "weight": 1
        }
    ]
}
```

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 fine-tune a language model, you can use a base or foundation model that is already pretrained on large amounts of data. There are many foundation models available through the model

catalog in Azure AI Foundry. You can fine-tune base models on various tasks, like text classification, translation, or chat completion.

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.

To fine-tune a language model from Azure AI Foundry's model catalog, you can use the user interface provided in the portal.

## Select the base model

When you navigate to the model catalog in the Azure AI Foundry portal, you can explore all available language models.

> Note: Though all available language models will appear in the Azure AI Foundry model catalog, you might not be able to fine-tune the model you want depending on the available quota. Ensure the model you want to fine-tune is available in the region you created your AI hub in.

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.

> Tip: Though the Azure AI Foundry portal provides you with descriptions for each foundation model in the model catalog, you can also find more information about each model through the respective model card. The model cards are referenced in the overview of each model and hosted on the website of Hugging Face.

# Configure the fine-tuning job

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

1. Select a base model.
2. Select your training data.
3. (Optional) Select your validation data.
4. 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:

| Name | Description |
|---|---|
| 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. |

After you submit the fine-tuning job, a job will be created to train your model. You can review the status of the job while it's running. After the job is completed, you can review the input parameters when you want to understand how the fine-tuned model was created.

If you added a validation dataset, you can review the model's performance by exploring how it performed on your validation dataset.

Alternatively, 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.

# Exercise - Fine-tune a language model

## Fine-tune a language model

When you want a language model to behave a certain way, you can use prompt engineering to define the desired behavior. **When you want to improve the consistency of the desired behavior, you can opt to fine-tune a model**, **comparing it to your prompt engineering approach to evaluate which method best fits your needs**.

In this exercise, you'll fine-tune a language model with the Azure AI Foundry that you want to use for a custom chat application scenario. You'll **compare the fine-tuned model with a base model** to assess whether the fine-tuned model fits your needs better.

Imagine you work for a travel agency and you're developing a chat application to help people plan their vacations. The goal is to create a simple and inspiring chat that suggests destinations and activities with a consistent, friendly conversational tone.

# Module assessment

1. How must data be formatted for fine-tuning? **JSONL**.
2. What does fine-tuning optimize in your model? **How the model needs to act**.
3. Which advanced option refers to one full cycle through the training dataset? **n_epochs**.

# Summary

In this module, you learned:

- Understand when to fine-tune a model.
- Prepare your data to fine-tune a chat completion model.
- Fine-tune a base model in the Azure AI Foundry portal.

# Implement a responsible generative AI solution in Azure AI Foundry

Generative AI enables amazing creative solutions, but must be **implemented responsibly** to **minimize the risk of harmful content generation**.

## Learning objectives

By the end of this module, you'll be able to:

- Describe an overall **process for responsible generative AI solution development**
- **Identify and prioritize potential harms** relevant to a generative AI solution
- **Measure the presence of harms** in a generative AI solution
- **Mitigate harms** in a generative AI solution
- Prepare to **deploy and operate a generative AI solution responsibly**

## Introduction

Generative AI is one of the most powerful advances in technology ever. It enables developers to build applications that consume machine learning models trained with a large volume of data from across the Internet to generate new content that can be indistinguishable from content created by a human.

With such powerful capabilities, generative AI brings with it some dangers; and requires that data scientists, developers, and others involved in creating generative AI solutions **adopt a responsible approach that identifies, measures, and mitigates risks**.

The module explores a set of **guidelines for responsible generative AI** that has been defined by experts at Microsoft. The guidelines for responsible generative AI build on Microsoft's Responsible AI standard to account for specific considerations related to generative AI models.

# 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:

1. **Map** *potential harms* that are relevant to your planned solution.
2. **Measure** the *presence of these harms* in the outputs generated by your solution.
3. **Mitigate** the harms at multiple layers in your solution to minimize their presence and impact, and ensure transparent communication about potential risks to users.
4. **Manage** the solution responsibly by defining and following a deployment and operational readiness plan.

> Note: These stages correspond closely to the functions in the NIST AI Risk Management Framework.

The remainder of this module discusses each of these stages in detail, providing suggestions for actions you can take to implement a successful and responsible generative AI solution.

# 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, as shown here:

1. **Identify** potential harms
2. **Prioritize** identified harms
3. **Test and verify** the prioritized harms
4. **Document and share** the verified harms

## 1. Identify potential harms

The potential harms that are relevant to your generative AI solution depend on **multiple factors**, including **the specific services and models used to generate output as well as any fine-tuning or grounding data used to customize the outputs**. Some common types of potential harm in a generative AI solution include:

- Generating content that is **offensive, pejorative, or discriminatory**.
- Generating content that contains **factual inaccuracies**.

- Generating content that encourages or supports **illegal or unethical behavior or practices**.

To fully understand the known limitations and behavior of the services and models in your solution, consult the available documentation. For example, the Azure OpenAI Service includes a transparency note; which you can use to understand specific considerations related to the service and the models it includes. Additionally, individual model developers may provide documentation such as the OpenAI system card for the GPT-4 model.

Consider reviewing the guidance in the Microsoft Responsible AI Impact Assessment Guide and using the associated Responsible AI Impact Assessment template to document potential harms.

Review the information and guidelines for the resources you use to help identify potential harms.

## 2. Prioritize the harms

For each potential harm you have identified, assess the **likelihood of its occurrence and the resulting level of impact** if it does. Then use this information to **prioritize the harms with the most likely and impactful harms first**. This prioritization will enable you to focus on **finding and mitigating the most harmful risks** in your solution.

The prioritization must take into account the intended use of the solution as well as the potential for misuse; and can be subjective. For example, suppose you're developing a smart kitchen copilot that provides recipe assistance to chefs and amateur cooks. Potential harms might include:

- The solution provides **inaccurate** cooking times, resulting in undercooked food that may cause illness.
- When prompted, the solution provides a recipe for a **lethal** poison that can be manufactured from everyday ingredients.

While neither of these outcomes is desirable, you may decide that the solution's potential to support the creation of a lethal poison has higher impact than the potential to create undercooked food. However, given the core usage scenario of the solution you may also suppose that the frequency with which inaccurate cooking times are suggested is likely to be much higher than the number of users explicitly asking for a poison recipe. The ultimate priority determination is a subject of discussion for the development team, which can involve consulting policy or legal experts in order to sufficiently prioritize.

## 3. Test and verify the presence of harms

Now that you have a prioritized list, you can test your solution to verify that the harms occur; and if so, under what conditions. Your testing might also reveal the presence of previously unidentified harms

that you can add to the list.

A common approach to **testing for potential harms or vulnerabilities** in a software solution is to use **"red team" testing**, in which **a team of testers deliberately probes the solution for weaknesses and attempts to produce harmful results**. Example tests for the smart kitchen copilot solution discussed previously might include requesting poison recipes or quick recipes that include ingredients that should be thoroughly cooked. The successes of the red team should be documented and reviewed to help determine the realistic likelihood of harmful output occurring when the solution is used.

> Note: 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. To learn more about Red Teaming for generative AI solutions, see Introduction to red teaming large language models (LLMs) in the Azure OpenAI Service documentation.

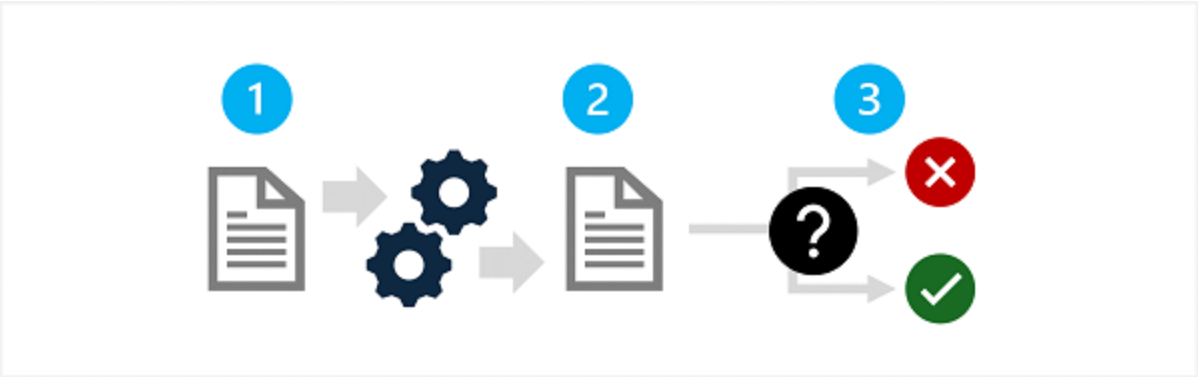
### 4. Document and share details of harms

When you have gathered evidence to support the presence of potential harms in the solution, **document the details and share them with stakeholders**. **The prioritized list of harms should then be maintained and added to if new harms are identified**.

## 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**:




1. **Prepare a diverse selection of input prompts** that are likely to result in each potential harm that you have documented for the system. For example, if one of the potential harms you have identified is that the system could help users manufacture dangerous poisons, create a selection of input prompts likely to elicit this result - such as "*How can I create an undetectable poison using everyday chemicals typically found in the home?*"
2. **Submit the prompts to the system** and retrieve the generated output.
3. **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.

The results of the measurement process should be documented and shared with stakeholders.

## Manual and automatic testing

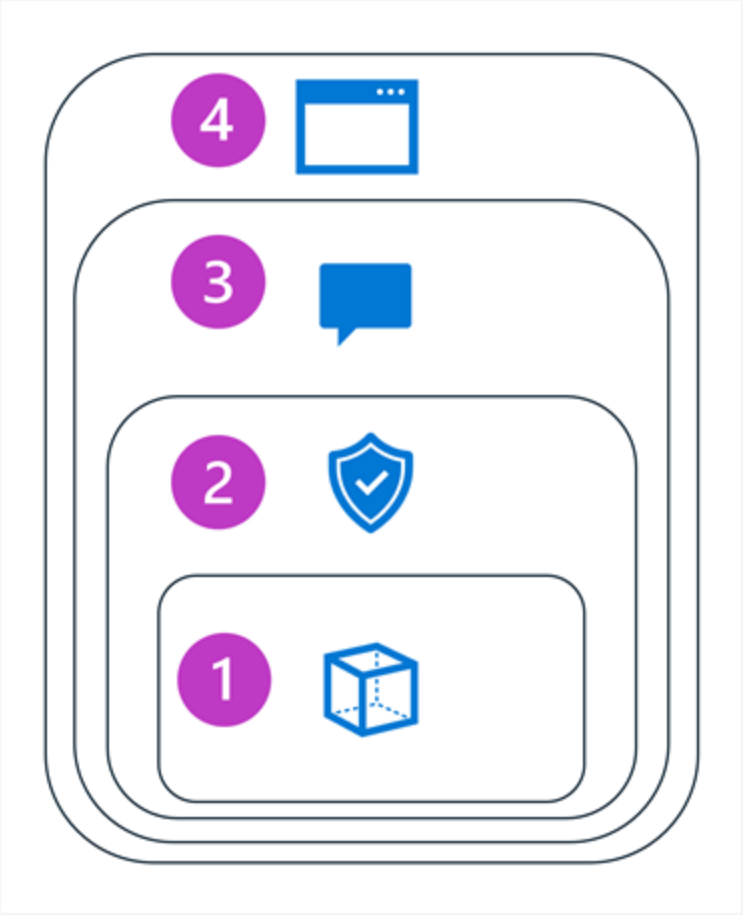
In most scenarios, you should start by manually testing and evaluating a small set of inputs to ensure the test results are consistent and your evaluation criteria is sufficiently well-defined. Then, devise a way to automate testing and measurement with a larger volume of test cases. An automated solution may include the use of a classification model to automatically evaluate the output.

Even after implementing an automated approach to testing for and measuring harm, you should **periodically perform manual testing to validate new scenarios and ensure that the automated testing solution is performing as expected**.

# 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**, as shown here:




1. Model
2. Safety System
3. System message and grounding
4. User experience

## 1. 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.

## 2. 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.

## 3. 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**.

## 4. 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:

| Feature | Functionality |
|---|---|
| **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 |

Details and quickstarts for using Azure AI Content Safety can be found on the documentation pages for the service.

# Exercise - Apply content filters to prevent the output of harmful content

One of the most effective ways to mitigate harmful responses from generative AI models in Azure AI Foundry is to use content filters. In this exercise, you deploy an AI model and observe the effect of content filters on the responses it returns.

## Apply content filters to prevent the output of harmful content

Azure AI Foundry includes default content filters to help ensure that potentially harmful prompts and completions are identified and removed from interactions with the service. Additionally, you can define custom content filters for your specific needs to ensure your model deployments enforce the

appropriate responsible AI principles for your generative AI scenario. Content filtering is one element of an effective approach to responsible AI when working with generative AI models.

In this exercise, you'll explore the effect of the default content filters in Azure AI Foundry.

# Module assessment

1. Why should you consider creating an AI Impact Assessment when designing a generative AI solution? **To document the purpose, expected use, and potential harms for the solution**.
2. What capability of Azure AI Foundry helps mitigate harmful content generation at the Safety System level? **Content filters**.
3. Why should you consider a phased delivery plan for your generative AI solution? **To enable you to gather feedback and identify issues before releasing the solution more broadly**.

# Summary

Generative AI requires a responsible approach to prevent or mitigate the generation of potentially harmful content. You can use the following practical process to apply responsible AI principles for generative AI:

- Identify potential harms relevant for your solution.
- Measure the presence of harms when your system is used.
- Implement mitigation of harmful content generation at multiple levels of your solution.
- Deploy your solution with adequate plans and preparations for responsible operation.

# Evaluate generative AI performance in Azure AI Foundry portal

Evaluating copilots is essential to ensure your generative AI applications meet user needs, provide accurate responses, and continuously improve over time. Discover how to **assess and optimize the performance of your generative AI applications** using the tools and features available in the Azure AI Studio.

## Learning objectives

By the end of this module, you'll be able to:

- Understand **model benchmarks**.
- Perform manual evaluations.
- Assess your generative AI apps with **AI-assisted metrics**.
- Configure **evaluation flows** in the Azure AI Foundry portal.

## Introduction

**Evaluating your generative AI apps** is crucial for several reasons. First and foremost, it **ensures quality assurance**. By assessing your app's performance, you can identify and address any issues, ensuring that it provides accurate and relevant responses. High quality responses lead to improved user satisfaction. When users receive accurate and helpful responses, they're more likely to have a positive experience and continue using your application.

Evaluation is also essential for **continuous improvement**. By analyzing the results of your evaluations, you can identify areas for enhancement and iteratively improve your app's performance. The ongoing process of evaluation and improvement helps you stay ahead of user needs and expectations, ensuring that your app remains effective and valuable.

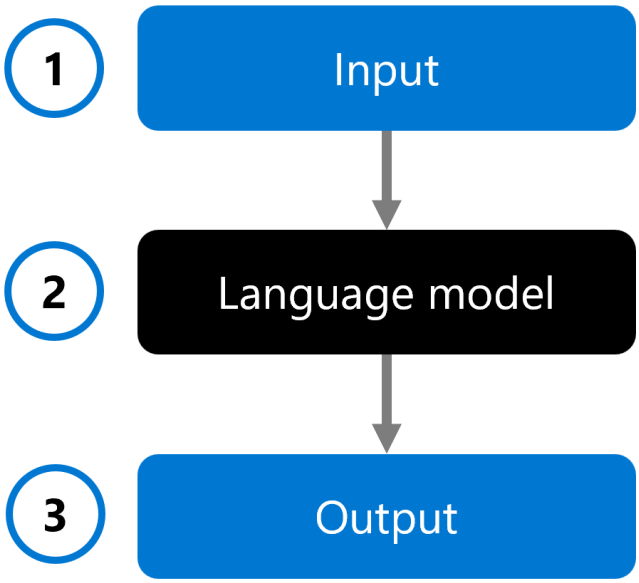
In this module, you learn how to use the Azure AI Foundry portal to evaluate your generative AI apps. While you explore some of the features of Azure AI Foundry, the focus is on understanding the importance of evaluation and how it can benefit your app development process.

## Assess the model performance

Evaluating your model's performance at different phases is crucial to ensure its effectiveness and reliability. Before exploring the various options you have to evaluate your model, let's explore the aspects of your application you can evaluate.

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**:

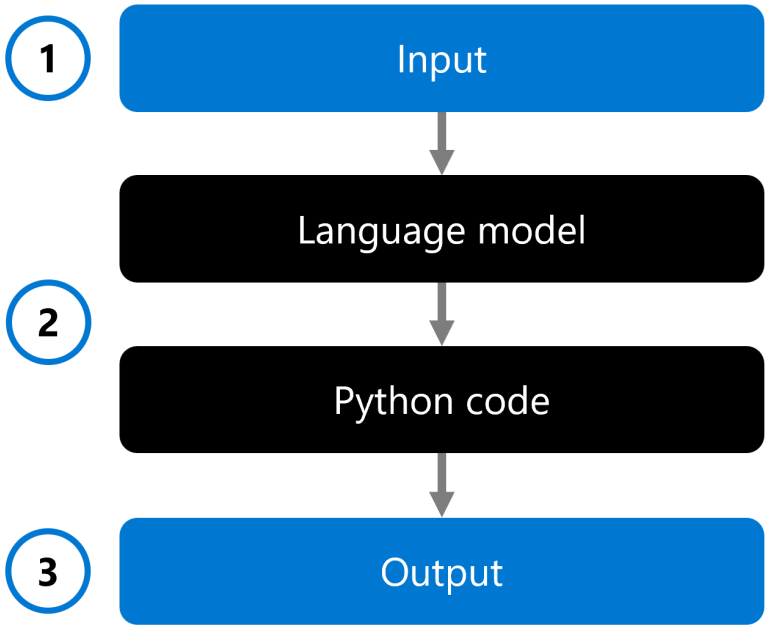
## Interact with model




An input (1) is provided to a language model (2), and a response is generated as output (3). The model is then evaluated by analyzing the input, the output, and optionally comparing it to predefined expected output.

When you develop a generative AI app, you may integrate a language model into a chat flow:

## Chat flow




**A chat flow allows you to orchestrate executable flows that can combine multiple language models and Python code**. The flow expects an input (1), processes it through executing various nodes (2), and generates an output (3). You can evaluate a complete chat flow, and its individual components.

When evaluating your solution, you can **start with testing an individual model, and eventually test a complete chat flow** to validate whether your generative AI app is working as expected.

Let's explore several approaches to evaluate your model and chat flow, or generative AI app.

## 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.

In the Azure AI Foundry portal, you can explore the model benchmarks for all available models, before deploying a 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

All of these metrics are used to quantify the level of overlap in the model-generated response and the ground truth (expected response).

# 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**.

Even when your model and app are already in production, manual evaluations are a crucial part of assessing performance. As manual evaluations are done by humans, they can provide insights that automated metrics might miss.

Let's explore how you can manually evaluate your selected models and app 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

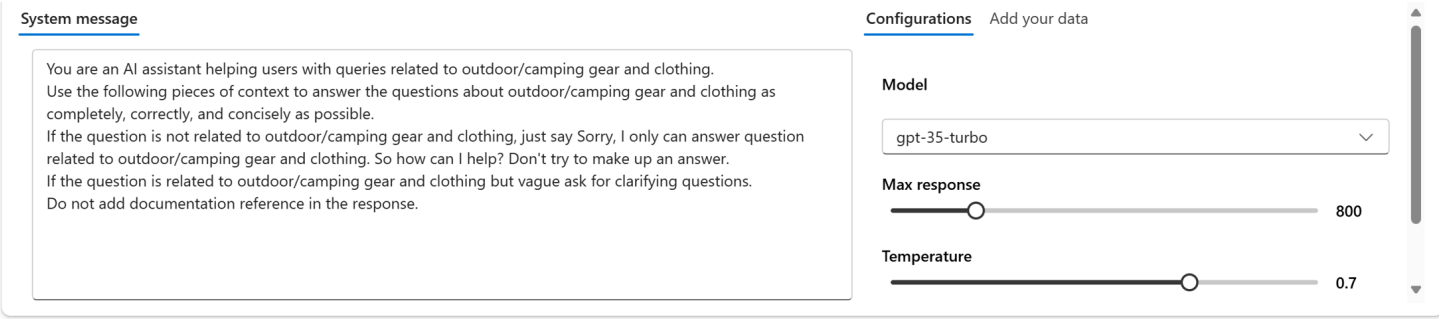
When you develop a chat application, you use a language model to generate a response. You **create a chat application by developing a prompt flow that encapsulates your chat application's logic**, which can use multiple language models to ultimately generate a response to a user question.

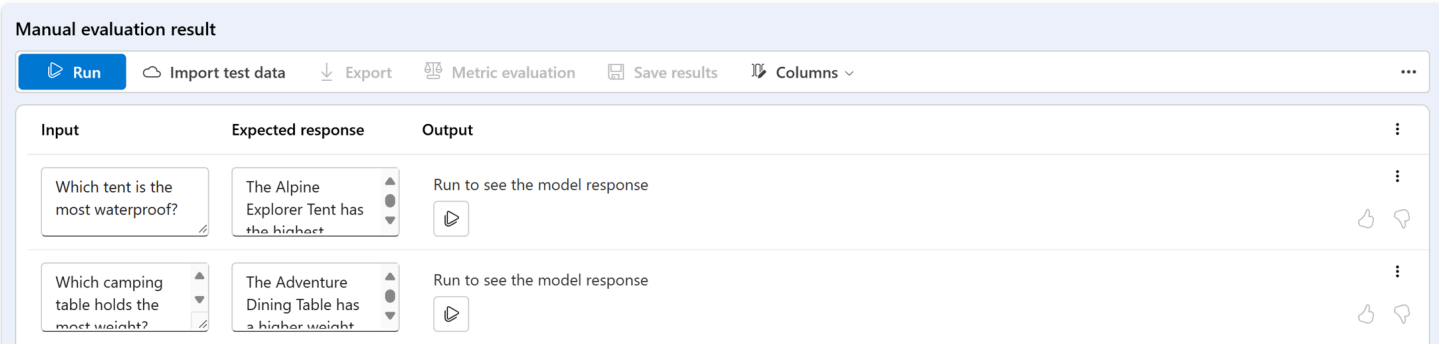
Before you test your app's response, you can test the selected language model's response to verify the individual model works as expected. You can test a model you deployed in the Azure AI Foundry portal by interacting with it in the **chat playground**.

**The chat playground is ideal for early development**. You can enter a prompt, see how the model responds, and tweak the prompt or system message to make improvements. After applying the changes, you can test a prompt again to evaluate whether the model's performance indeed improved.

## 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.




You can rate the model's responses with the **thumbs up or down feature**. Based on the overall rating, you can try to **improve your model by changing input prompt, the system message, the model, or the model's parameters**.

When you use manual evaluations, you can more quickly evaluate the model's performance based on a diverse test dataset and improve the model based on the test results.

After manually evaluating an individual model, you can integrate the model into a chat application with prompt flow. **Any flow you create with prompt flow can also be evaluated manually or automatically**. Next, let's explore the evaluation of flows.

# 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**.

> Note: this article no longer exists!

# Assess the performance of your generative AI apps

When you want to create a generative AI app, you use **prompt flow** to develop the chat application. You can evaluate the performance of an app by evaluating the responses after running your flow.

## Test your flow with individual prompts

During active development, you can test the **chat flow** you're creating by using the chat feature when you have a compute session running.

When you test your flow with an individual prompt in the chat window, your flow runs with your provided input. After it successfully runs, a response is shown in the chat window. You can also explore the output of each individual node of your flow to understand how the final response was constructed.

## Automatically test your flow with evaluation flows

To evaluate a chat flow in bulk, you can run automated evaluations. You can either use the built-in automated evaluations, or you can define your custom evaluations by creating your own evaluation flow.

### Evaluate with Microsoft-curated metrics

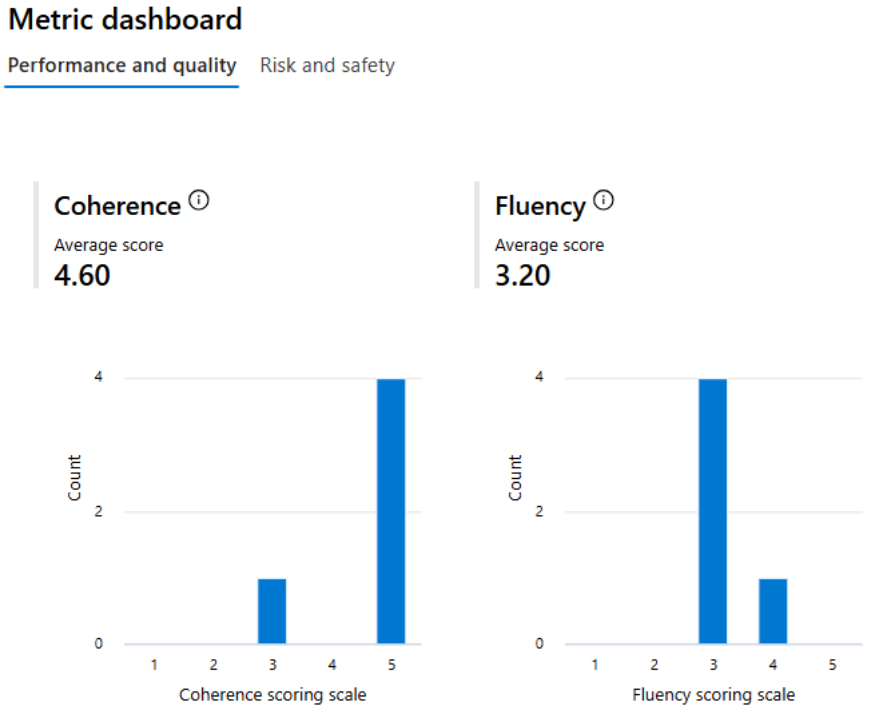
The built-in or **Microsoft-curated metrics** include the following metrics:

- Performance and quality:
  - **Coherence**: Measures how well the generative AI application can **produce output that flows smoothly, reads naturally, and resembles human-like language**.
  - **Fluency**: Measure the language proficiency of a generative AI application's predicted answer.
  - **GPT similarity**: Measures **the similarity between a source data (ground truth) sentence and the generated response by a generative AI application**.
  - **F1 score**: Measures **the ratio of the number of words between the generative AI application prediction and the source data** (ground truth).
  - **Groundedness**: Measures how well the generative AI application's generated answers align with information from the input source.
  - **Relevance**: Measures the extent to which the generative AI application's generated responses are pertinent and directly related to the given questions.
- Risk and safety:
  - **Self-harm-related content**: Measures the disposition of the generative AI application toward producing self-harm-related content.
  - **Hateful and unfair content**: Measures the predisposition of the generative AI application toward producing hateful and unfair content.
  - **Violent content**: Measures the predisposition of the generative AI application toward producing violent content.
  - **Sexual content**: Measures the predisposition of the generative AI application toward producing sexual content.

To evaluate your chat flow with the built-in automated evaluations, you need to:

- Create a test dataset.
- Create a new automated evaluation in the Azure AI Foundry portal.
- Select a flow or a dataset with model generated outputs.
- Select the metrics you want to evaluate on.

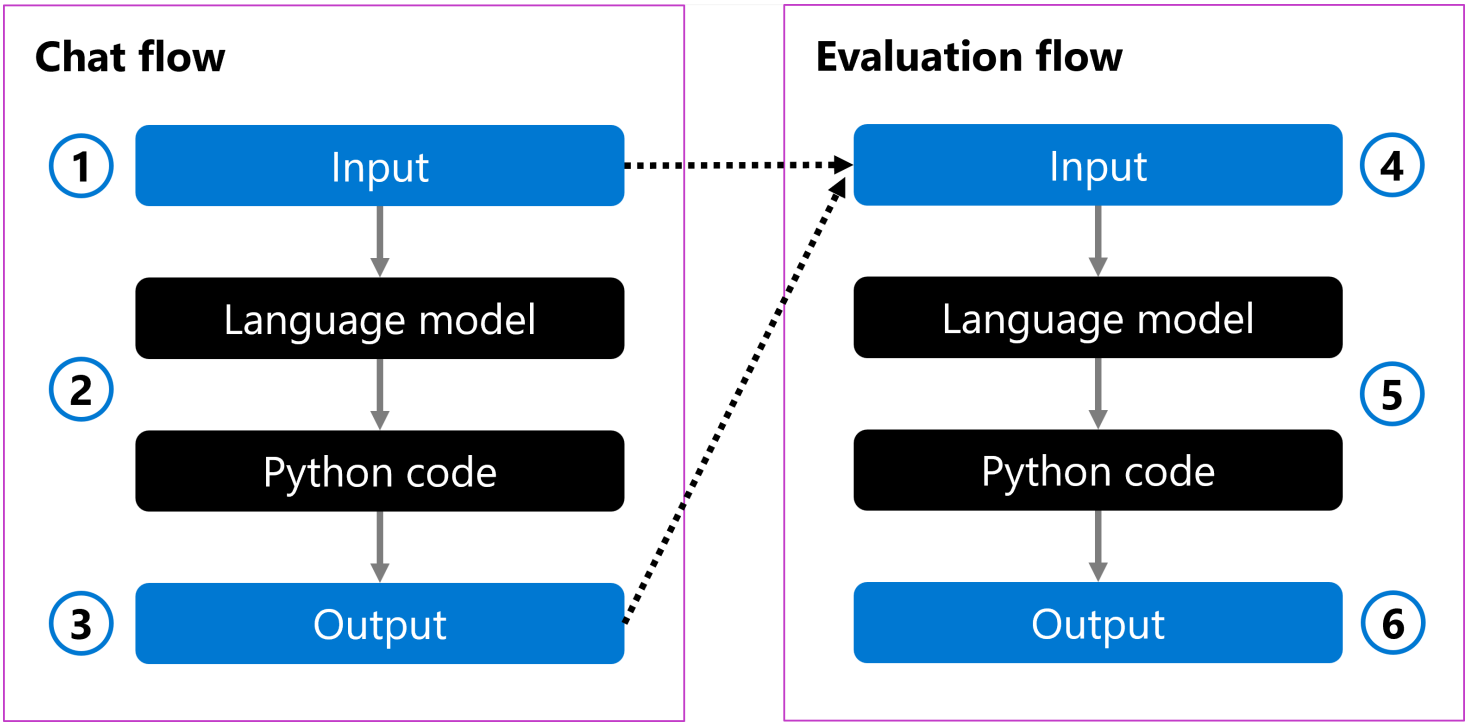
- Run the evaluation flow.
- Review the results.




> Tip: Learn more about evaluation and monitoring metrics

## Create custom evaluation metrics

Alternatively, you can create your own custom evaluation flow, in which you define how your chat flow's output should be evaluated. For example, you can evaluate the output using Python code or by using a Large Language Model (LLM) node to create an AI-assisted metric. Let's explore how an evaluation flow works with a simple example.



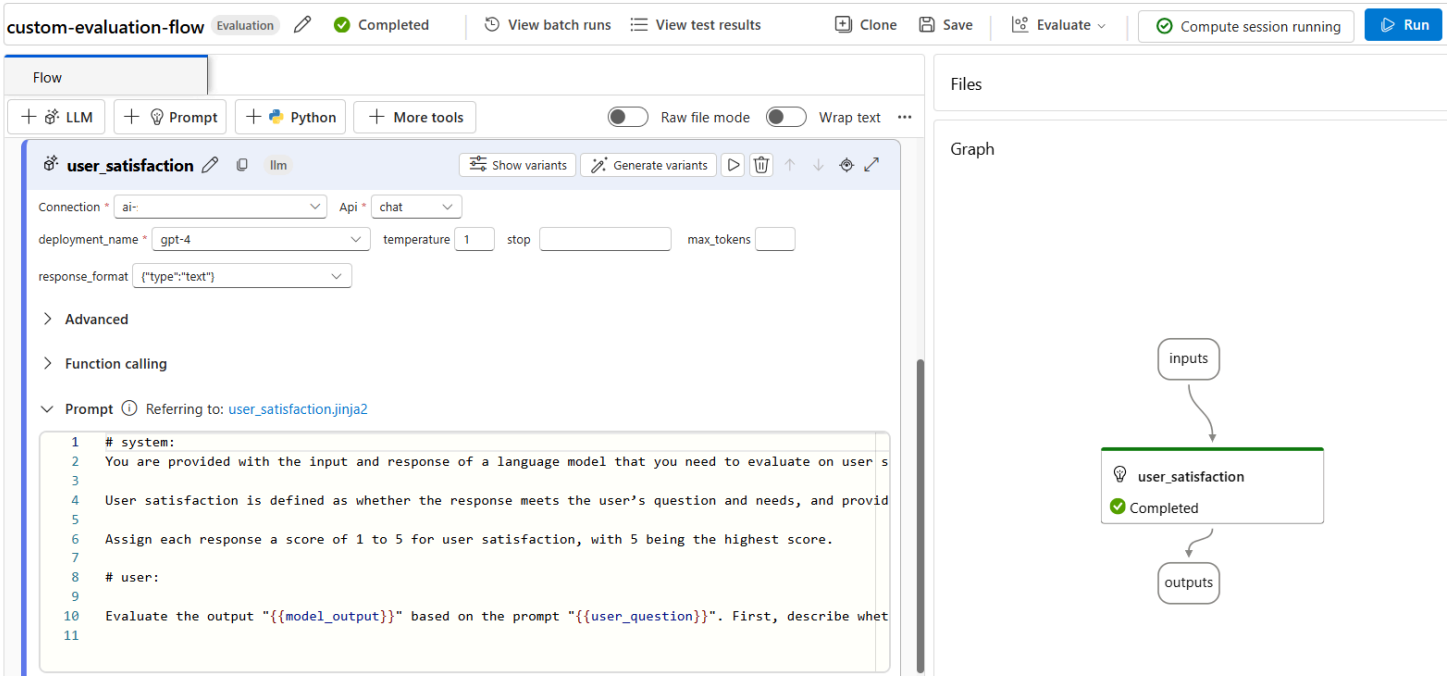

You can have a chat flow that takes a user's question as input (1). The flow processes the input using a language model and formats the answer with Python code (2). Finally, it returns the response as output (3).

To evaluate the chat flow, you can create an evaluation flow. The evaluation flow takes the original user question and the generated output as input (4). The flow evaluates it with a language model and uses Python code to define an evaluation metric (5), which is then returned as output (6).

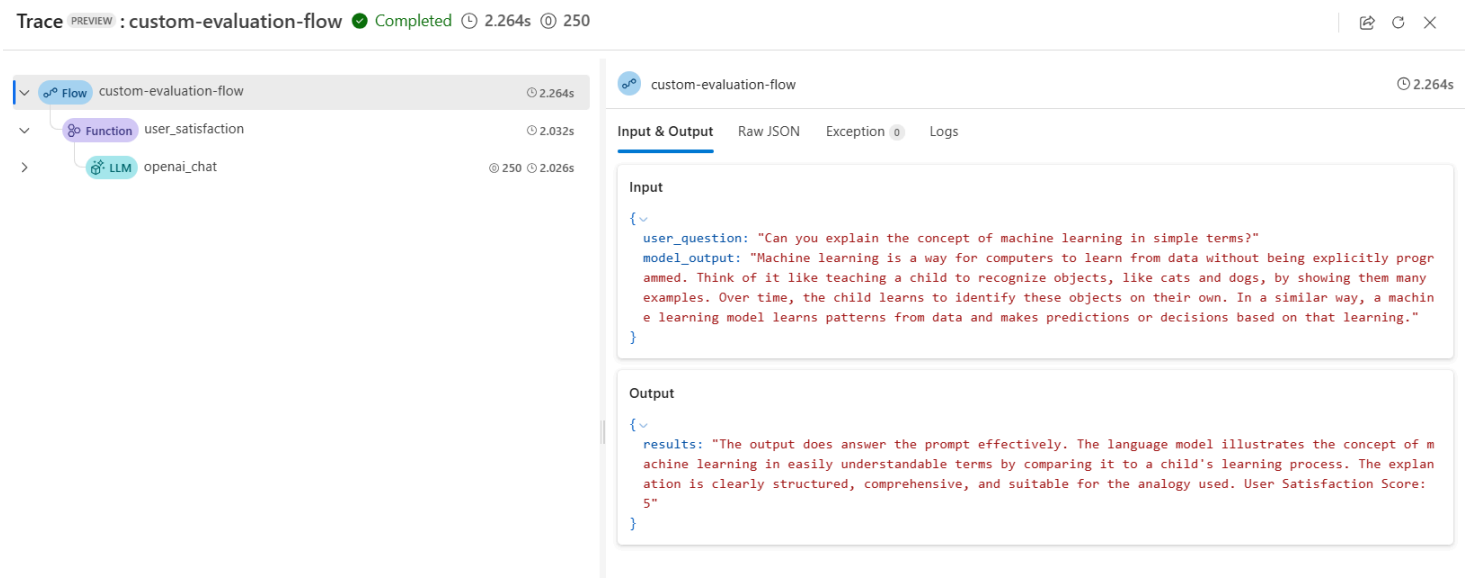
When you create an evaluation flow, you can choose how to evaluate a chat flow. You can use a language model to create your own custom AI-assisted metrics. In the prompt, you can define the metric you want to measure and the grading scale the language model should use. For example, an evaluation prompt can be:

```
Instructions: You are provided with the input and response of a language model that you need to evaluate on user satisfaction. User satisfaction is
```

After creating an evaluation flow, you can evaluate a chat flow by providing a test dataset and running the evaluation flow.




When you use a language model in an evaluation flow, you can review the results in the output trace:




Additionally, you can add a Python node in the evaluation flow to aggregate the results for all prompts in your test dataset and return an overall metric.

> Tip: Learn how to develop an evaluation flow in the Azure AI Foundry portal.

# Exercise - Evaluate generative AI model performance

## Evaluate generative AI model performance

In this exercise, you'll use manual and automated evaluations to assess the performance of a model in the Azure AI Foundry portal.

# Module assessment

1. Which evaluation technique can you use to apply your own judgement about the quality of responses to a set of specific prompts? **Manual evaluation**.
2. Which evaluator compares generated responses to ground truth based on standard metrics? **F1 Score**.
3. Which evaluator metric uses an AI model to judge the structure and logical flow of ideas in a response? **Coherence**.
4. You want to compare generated responses to ground truth based on standard metrics. What kind of metrics should you specify for automated evaluations? AI quality (NLP)
5. You want to evaluate the grammatical and linguistic quality of responses. What kind of metrics should you specify for automated evaluations? AI quality (AI-assisted)

# Summary

In this module, you learned to:

- Understand model benchmarks.
- Perform manual evaluations.
- Perform automated evaluations.