

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 your 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. Embedding 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 your 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.

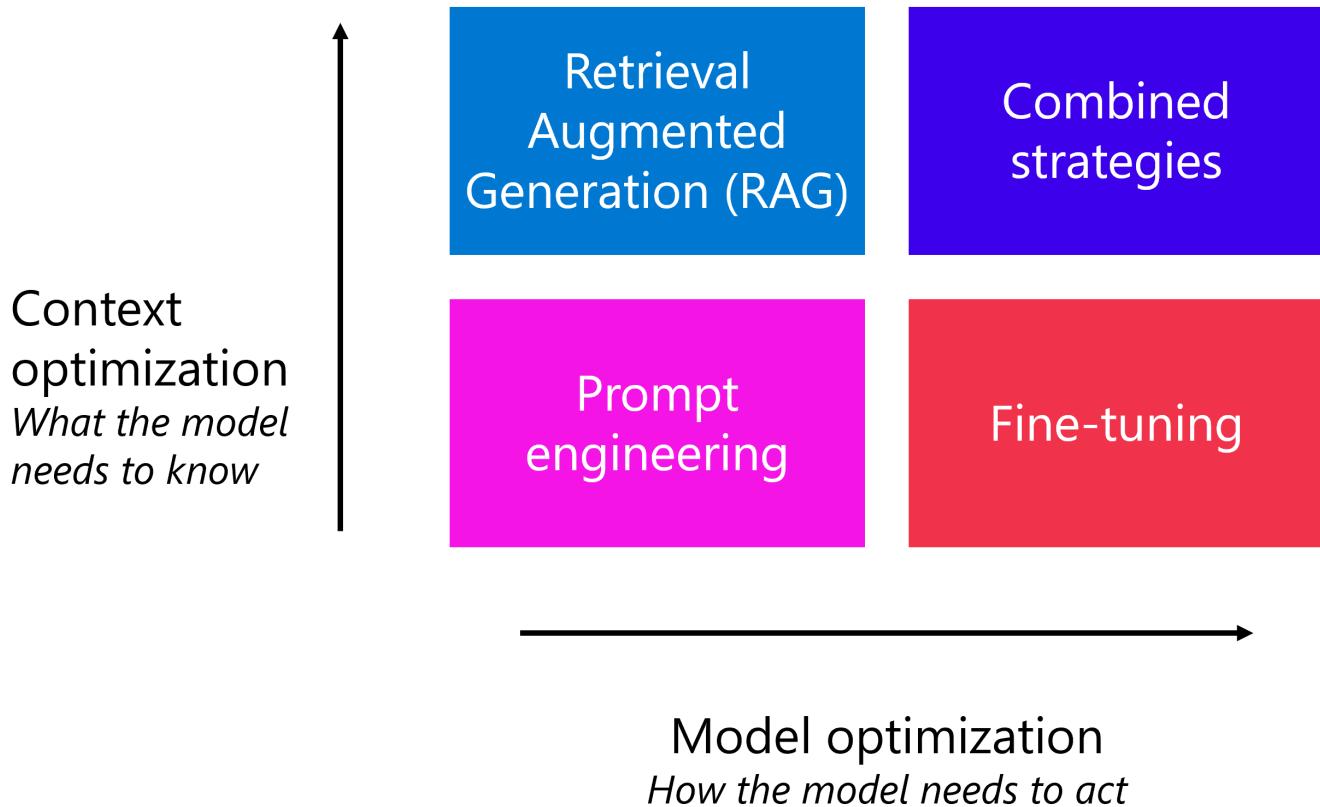
Prompt engineering can be an effective way to optimize model responses, but in some cases it may not provide sufficient context or guidance to always meet your exact needs. As a developer, you can consider the following additional optimization strategies to improve the relevance of your generative AI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

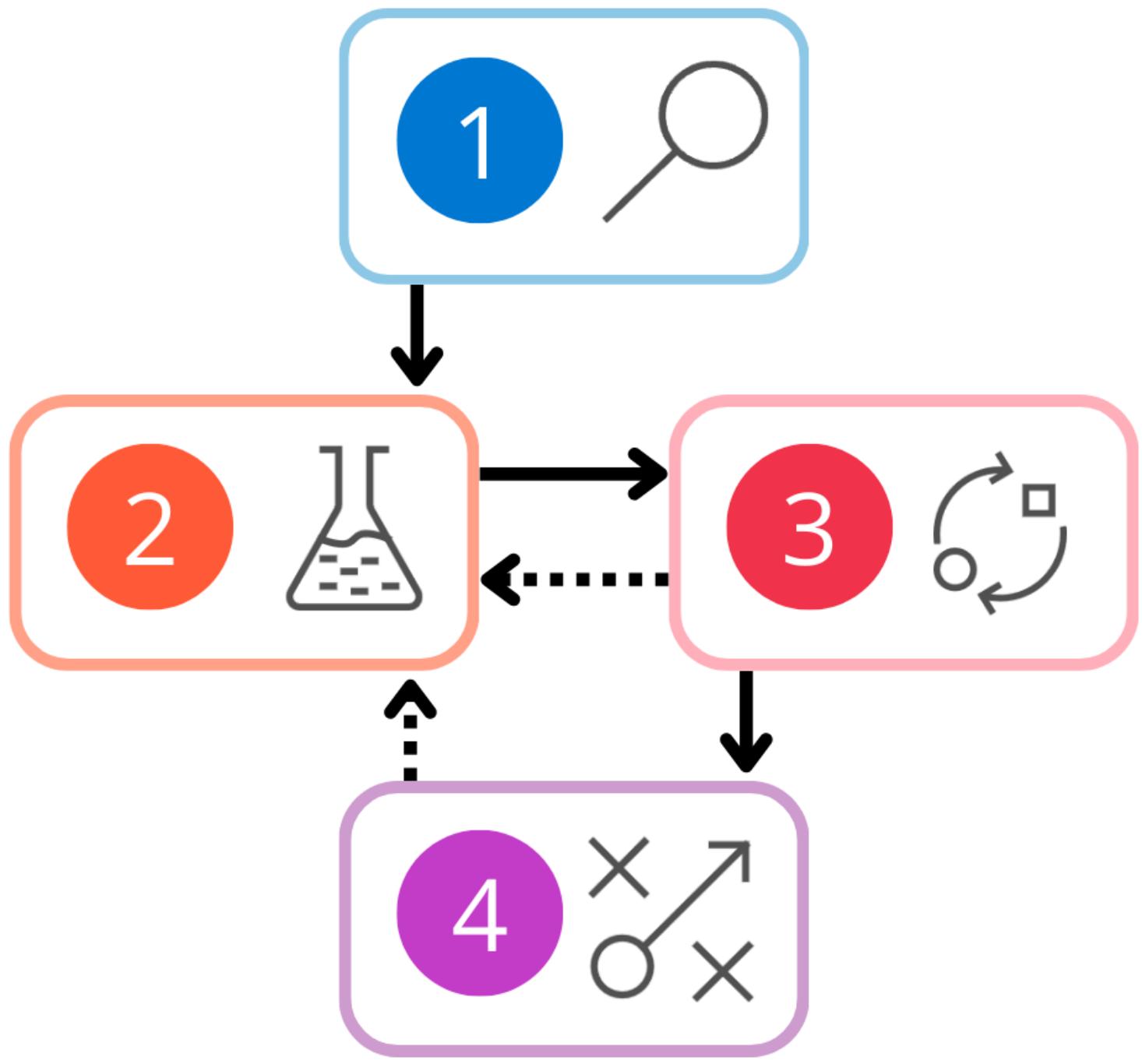
Understand the development lifecycle of a large language model (LLM) app

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.

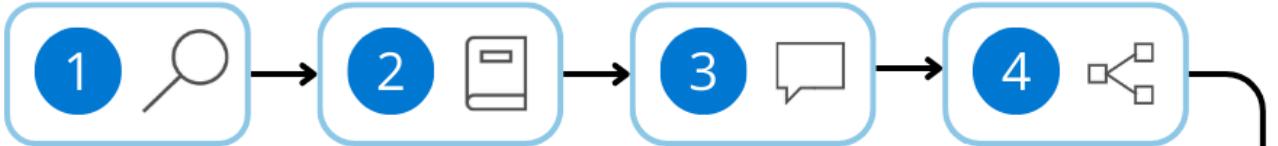
During production, you:

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.

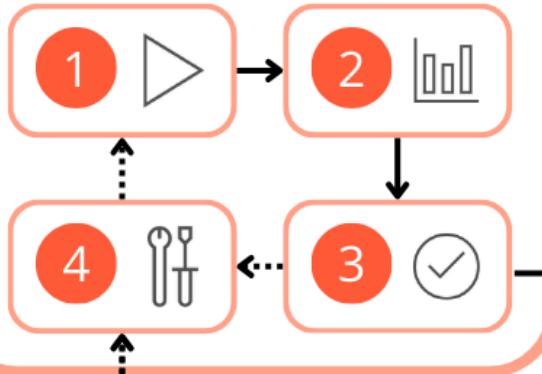
Explore the complete development lifecycle

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

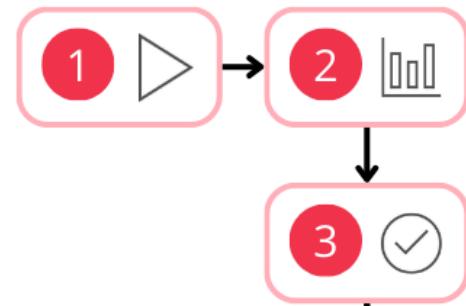
Initialization



Experimentation



Evaluation & refinement



Production



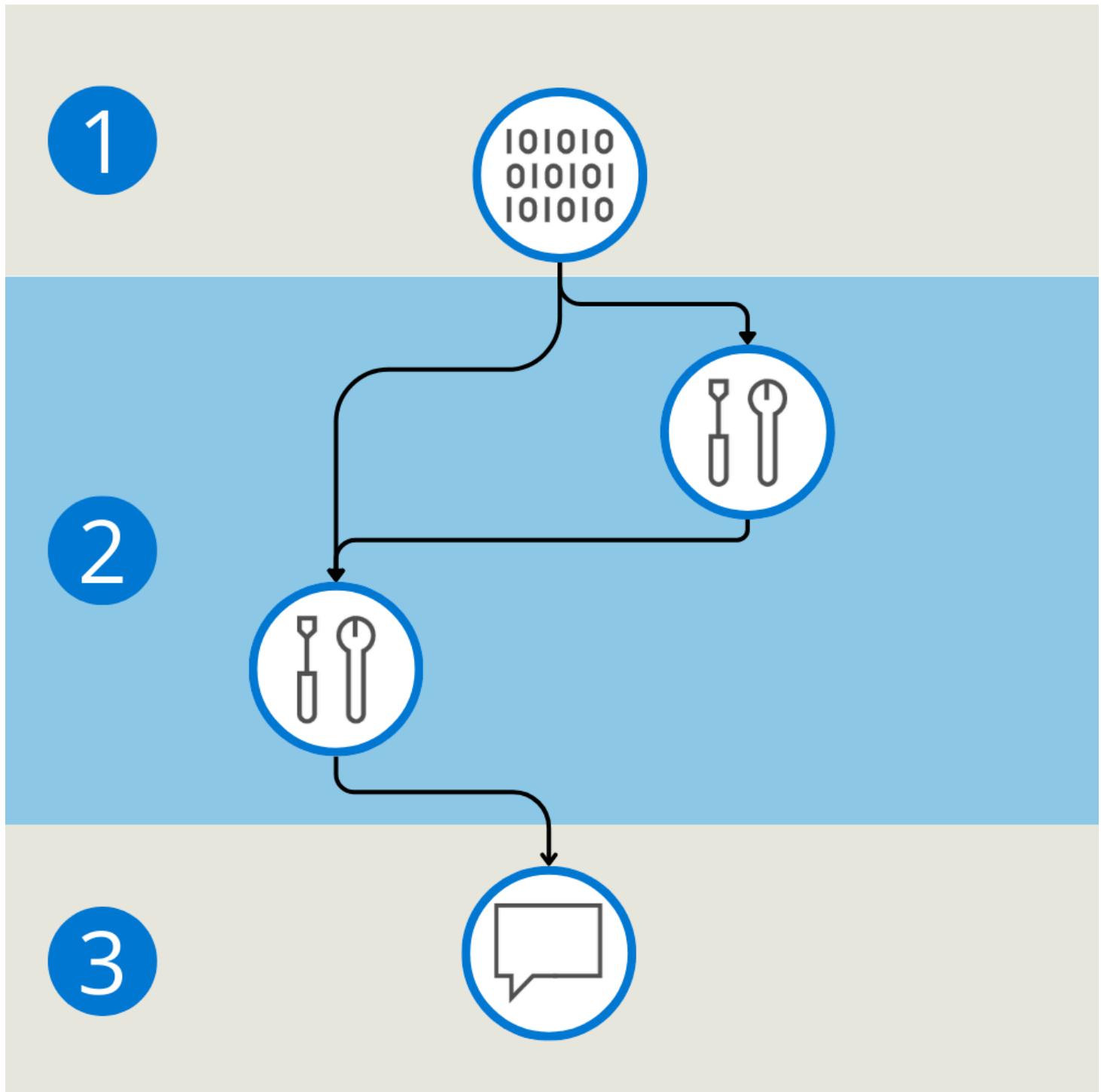
Understand core components and explore flow types

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.

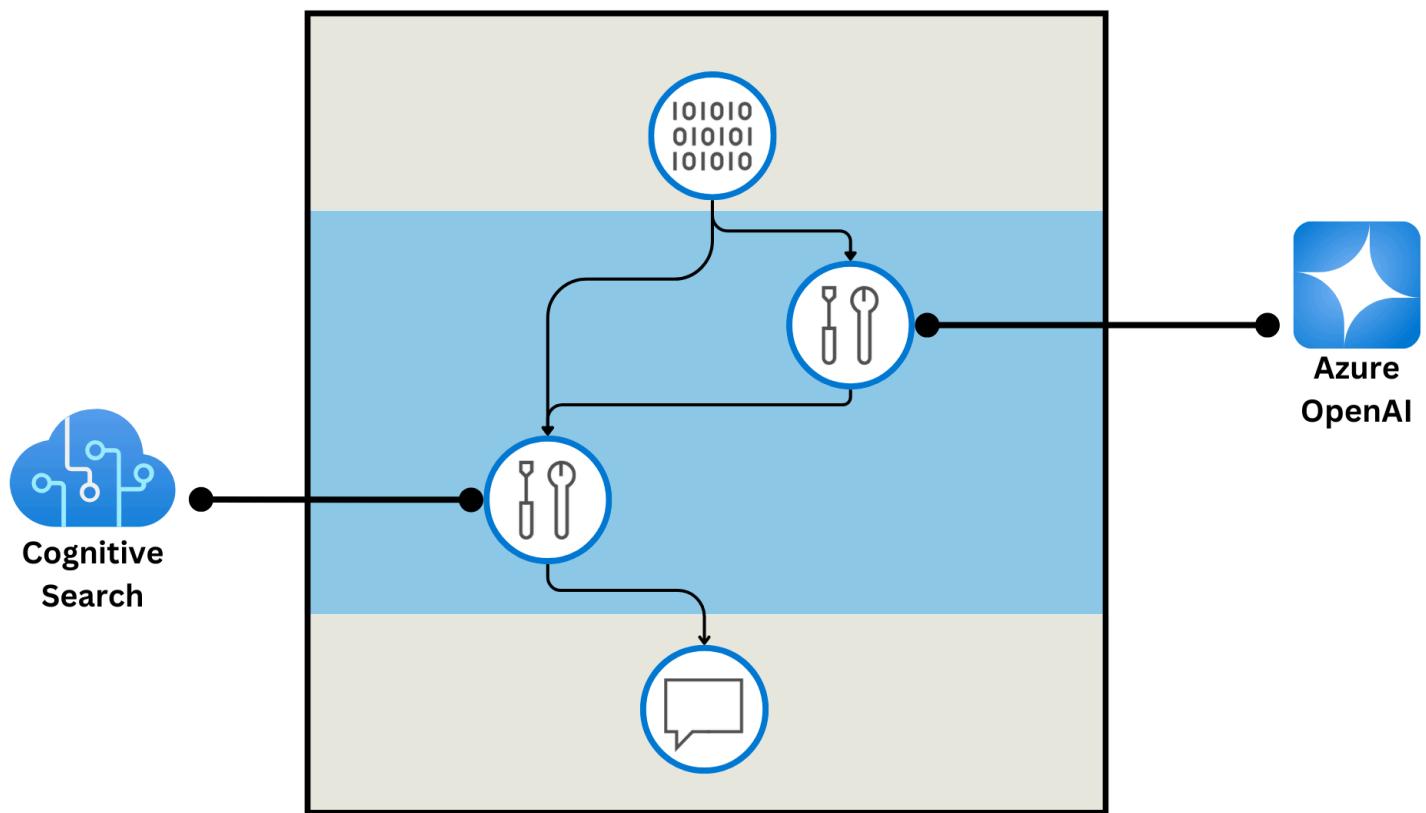
Now that you understand how a flow is structured and what you can use it for, let's explore how you can create a flow.

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:

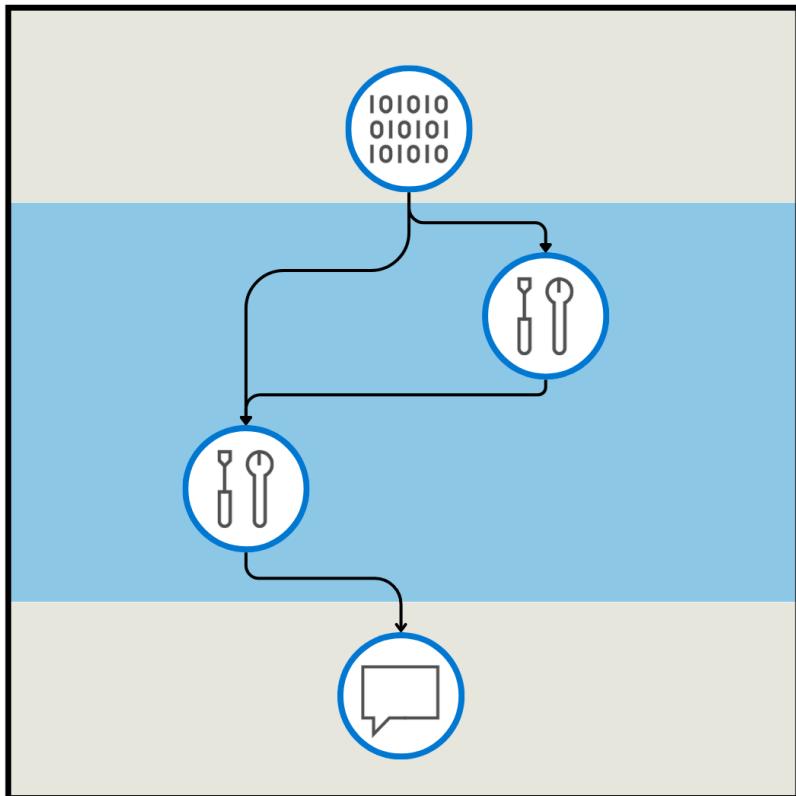
Connection type	Built-in tools
Azure OpenAI	LLM or Python

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.



1 ➤



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.

The screenshot shows the Azure AI Foundry interface for a project named 'Project50182944'. The left sidebar has a 'Prompt flow' section selected. The main area shows a 'Travel-Chat' flow. Inputs include 'chat_1' (list) and 'quest' (string). The output is 'chat'. The prompt template is:

```
1 # system:  
2 You are a helpful assistant.  
3  
4 {% for item in chat_history %}
```

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**.

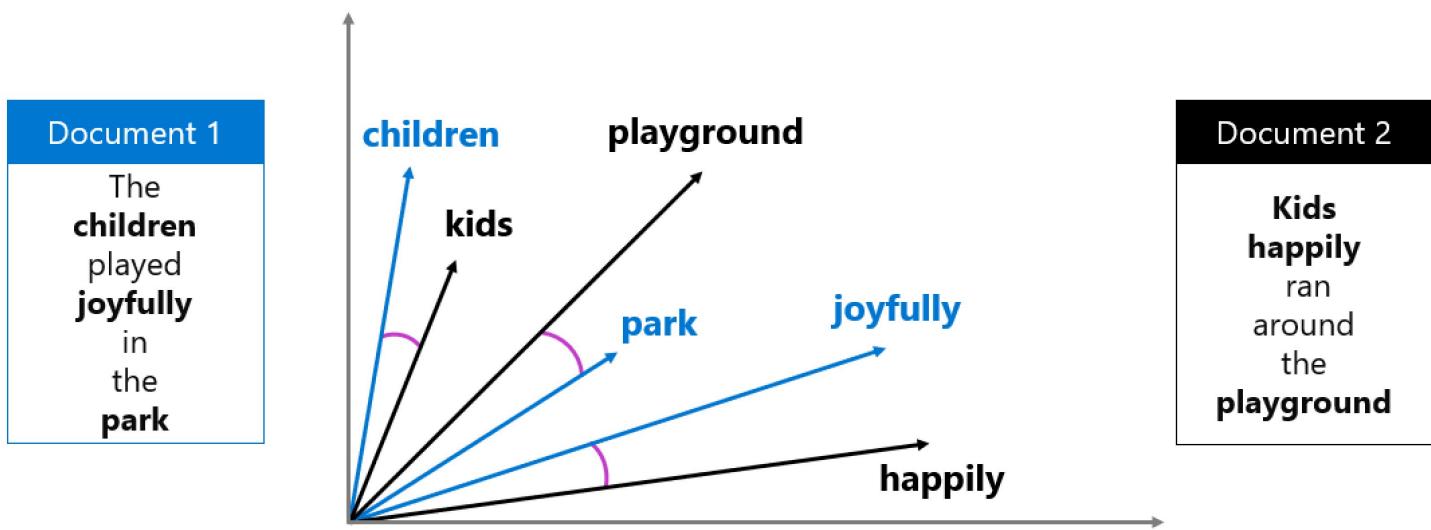
An **embedding** is a special format of data representation that a search engine can use to easily find the relevant information. More specifically, **an embedding is a vector of floating-point numbers**.

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-1
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.

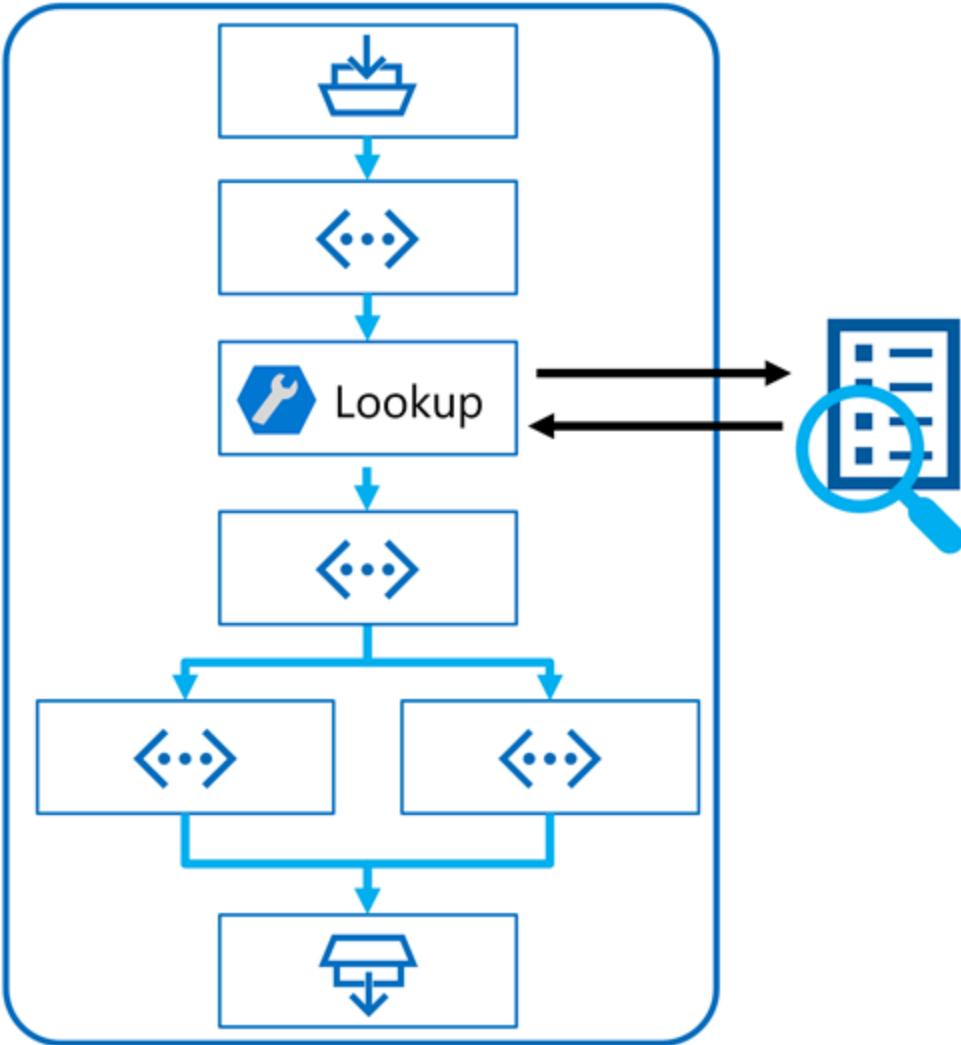
The flow is then defined as a series of connected tools, each of which performs a specific operation on the inputs and other environmental variables. There are multiple types of tool that you can include in a prompt flow to perform tasks such as:

- 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

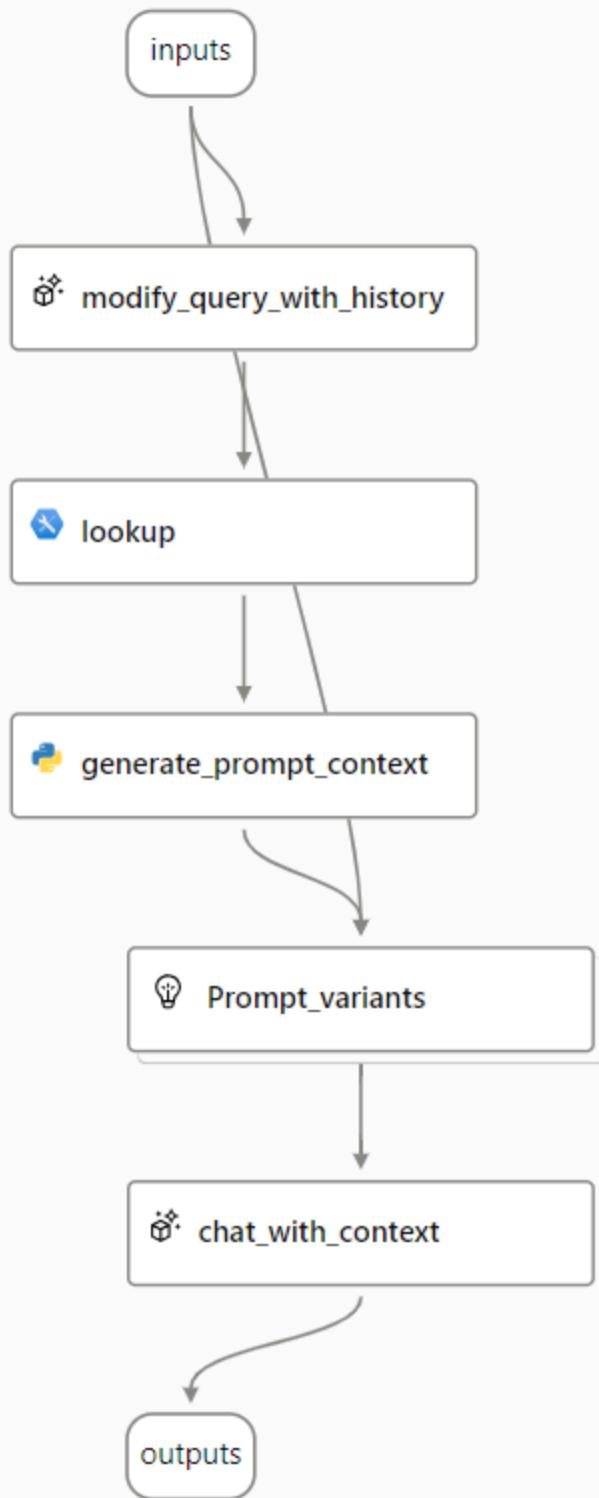
The key to using the RAG pattern in a prompt flow is to use **an Index Lookup tool to retrieve data from an index so that subsequent tools in the flow can use the results to augment the prompt used to generate output from an LLM.**



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.

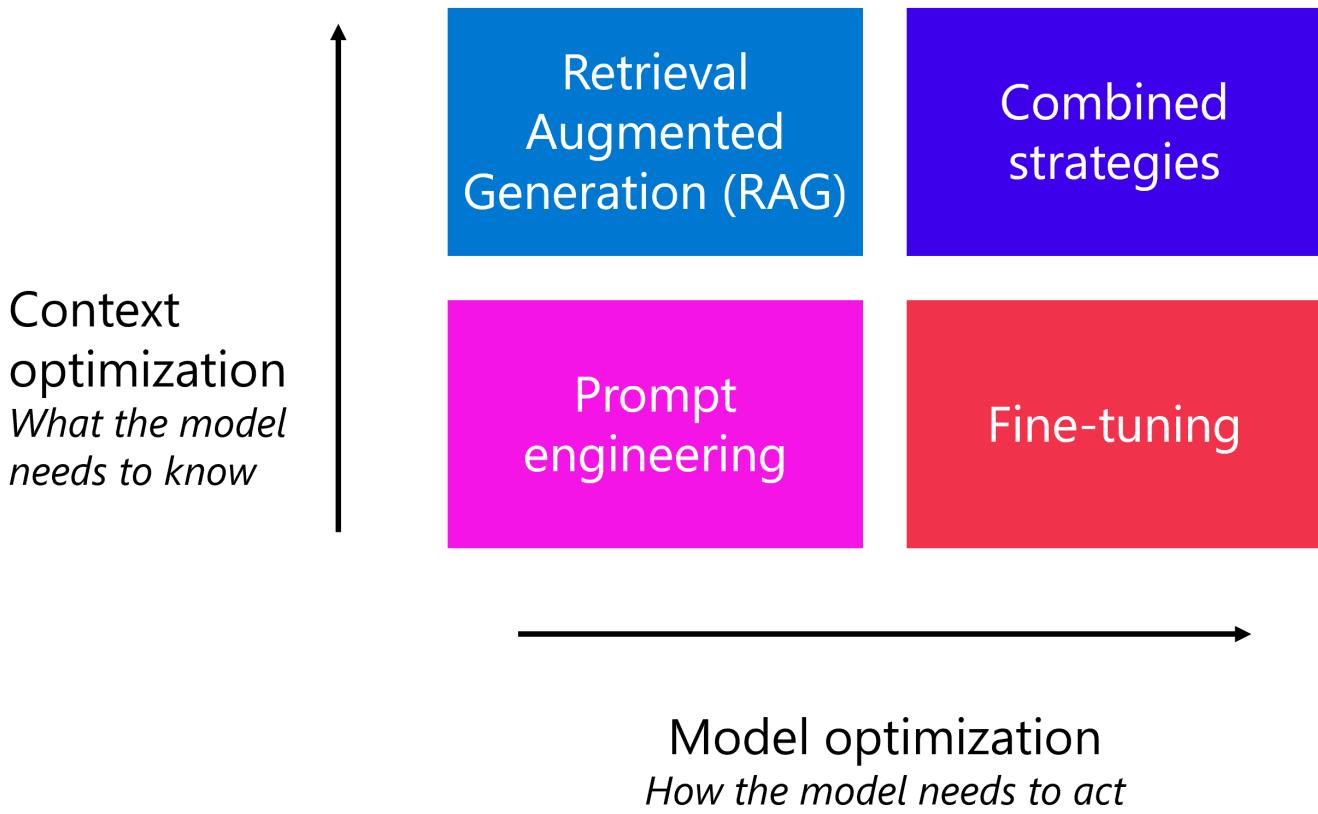
Understand when to fine-tune a language model

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 primary goal is to help users troubleshoot their Xbox consoles."}, {"role": "user", "content": "Is Xbox better than PlayStation?"}, {"role": "assistant", "content": "I apologize, but I cannot provide personal opinions. My primary job is to help users troubleshoot their Xbox consoles."}]}
```

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

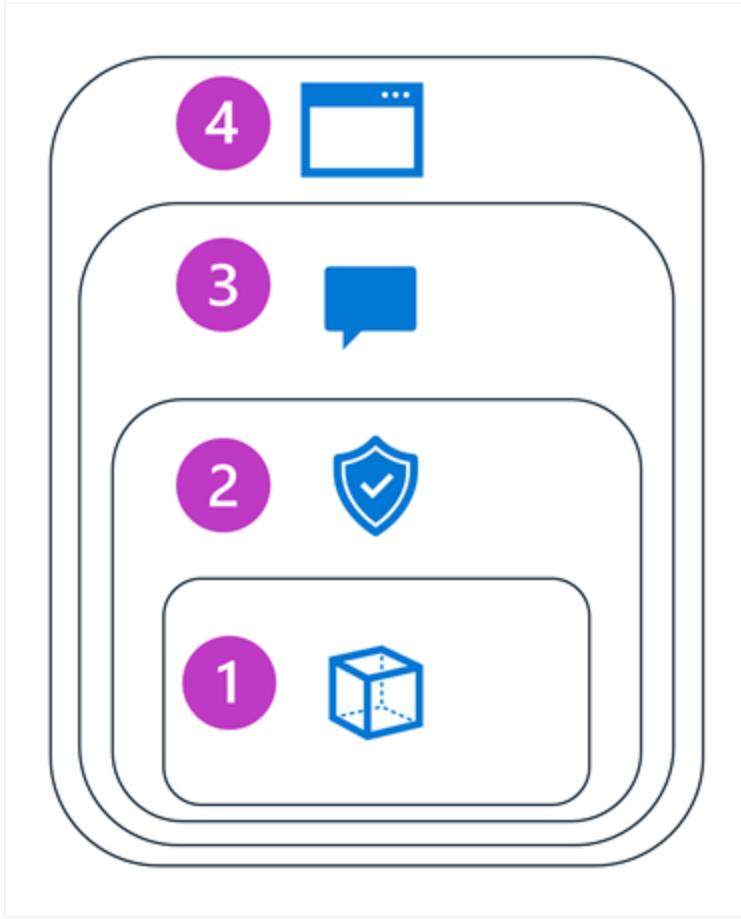
In most scenarios, you should start by manually testing and evaluating a small set of inputs to ensure the test results are consistent and your evaluation criteria is sufficiently well-defined. Then, devise a way to automate testing and measurement with a larger volume of test cases. An automated solution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

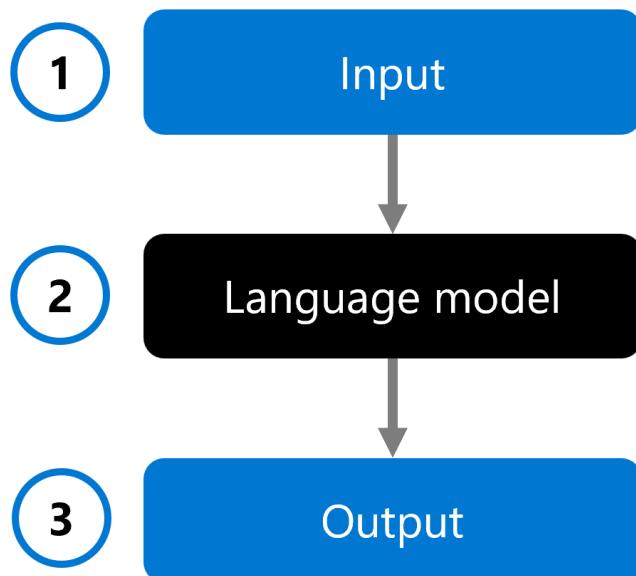
In this module, you learn how to use the Azure AI Foundry portal to evaluate your generative AI apps. While you explore some of the features of Azure AI Foundry, the focus is on understanding the importance of evaluation and how it can benefit your app development process.

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**:

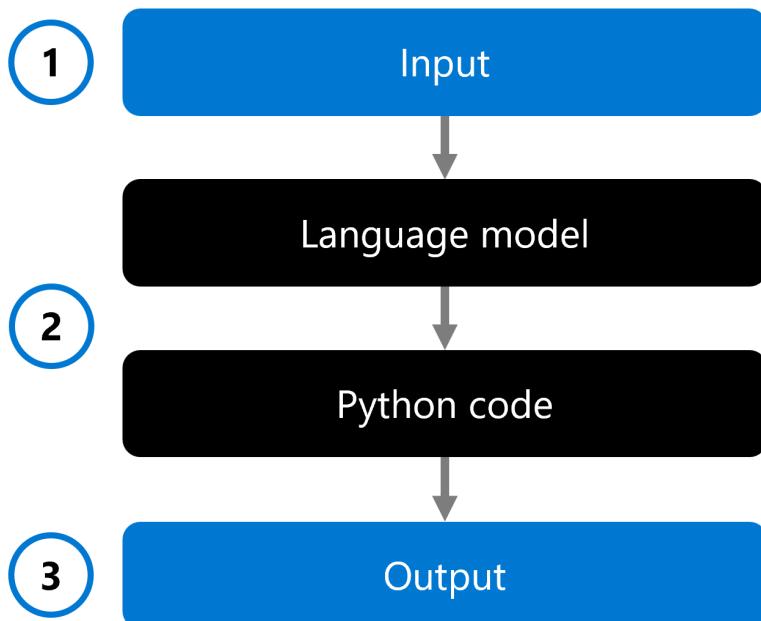
Interact with model



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.

The screenshot shows the Azure AI Foundry Chat Playground interface. On the left, under 'System message', there is a box containing a system message about being an AI assistant for outdoor/camping gear and clothing, along with some context and instructions. On the right, under 'Configurations', there are settings for the 'Model' (set to 'gpt-35-turbo'), 'Max response' (set to 800), and 'Temperature' (set to 0.7). Below these, the 'Manual evaluation result' section is shown, featuring a table with two rows of input, expected response, and output. The first row has an input of "Which tent is the most waterproof?", an expected response of "The Alpine Explorer Tent has the highest", and an output button labeled "Run to see the model response". The second row has an input of "Which camping table holds the most weight?", an expected response of "The Adventure Dining Table has a higher weight", and an output button labeled "Run to see the model response". Each row also has a thumbs up and thumbs down icon for rating.

Input	Expected response	Output
Which tent is the most waterproof?	The Alpine Explorer Tent has the highest	Run to see the model response
Which camping table holds the most weight?	The Adventure Dining Table has a higher weight	Run to see the model response

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.

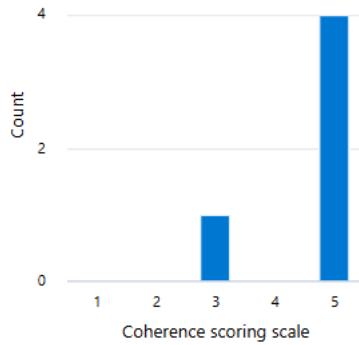
- Run the evaluation flow.
- Review the results.

Metric dashboard

Performance and quality Risk and safety

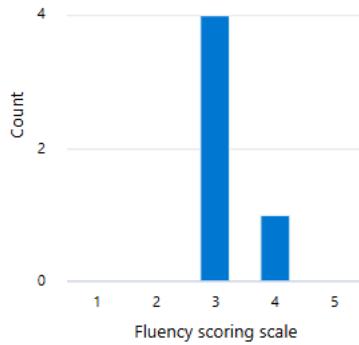
Coherence ⓘ

Average score
4.60



Fluency ⓘ

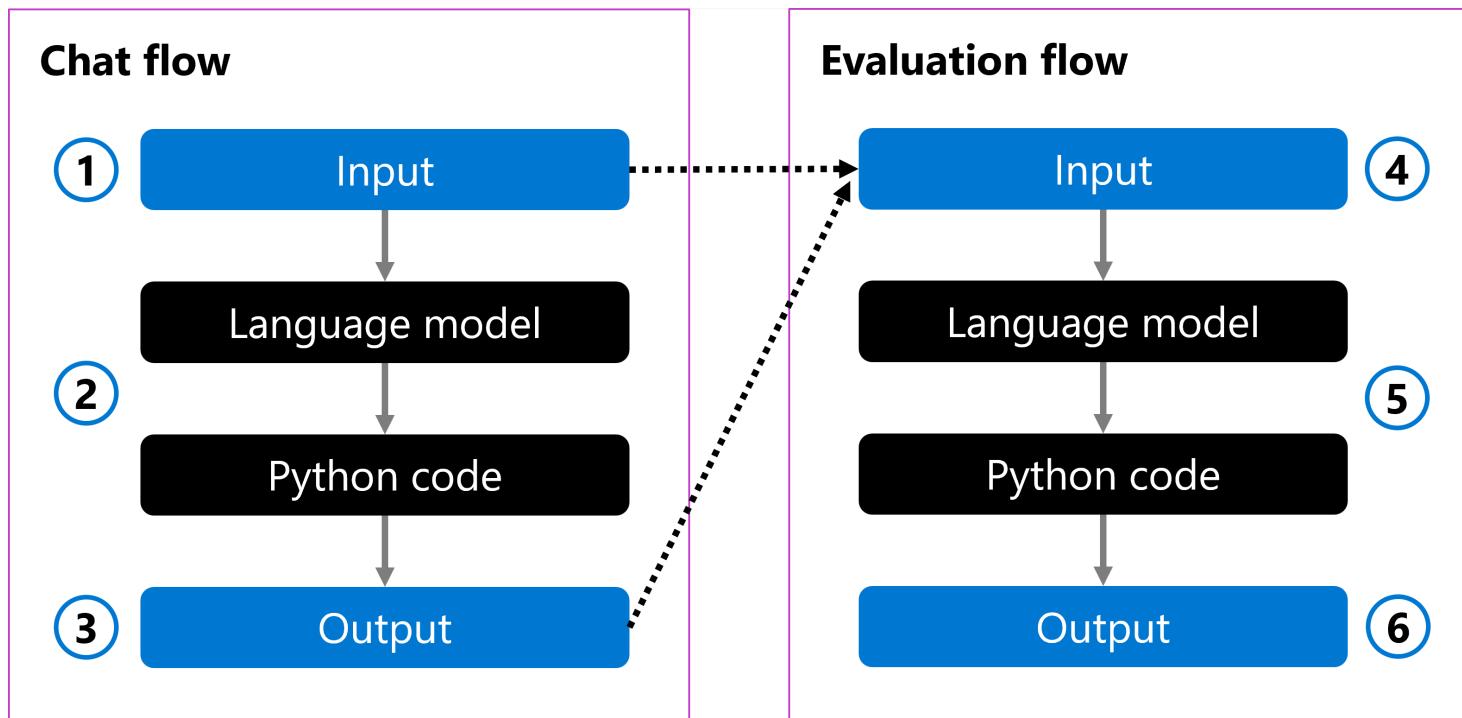
Average score
3.20



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.

```
graph TD; inputs((inputs)) --> user_satisfaction["user_satisfaction<br>Completed"]; user_satisfaction --> outputs((outputs))
```

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

```
user_question: "Can you explain the concept of machine learning in simple terms?"  
model_output: "Machine learning is a way for computers to learn from data without being explicitly programmed. Think of it like teaching a child to recognize objects, like cats and dogs, by showing them many examples. Over time, the child learns to identify these objects on their own. In a similar way, a machine learning model learns patterns from data and makes predictions or decisions based on that learning."
```

```
results: "The output does answer the prompt effectively. The language model illustrates the concept of machine learning in easily understandable terms by comparing it to a child's learning process. The explanation is clearly structured, comprehensive, and suitable for the analogy used. User Satisfaction Score: 5"
```

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.

Get started with AI agent development on Azure

AI agents represent the next generation of intelligent applications. Learn how they can be developed and used on Microsoft Azure.

Learning objectives

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

- Describe core concepts related to AI agents
- Describe options for agent development
- Create and test an agent in the Azure AI Foundry portal

Introduction

As generative AI models become more powerful and ubiquitous, their use has grown beyond simple "chat" applications to power intelligent agents that can operate autonomously to automate tasks.

Increasingly, organizations are **using generative AI models to build agents** that orchestrate business processes and coordinate workloads in ways that were previously unimaginable.

This module discusses some of the **core concepts related to AI agents**, and introduces some of the technologies that developers can use to build agentic solutions on Microsoft Azure.

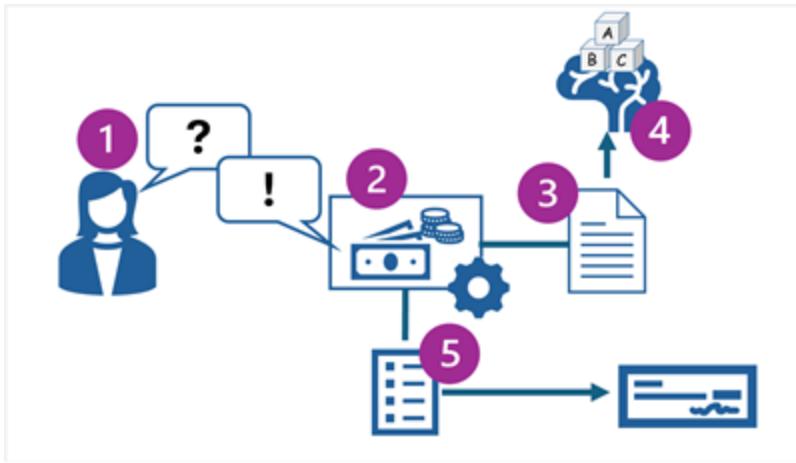
What are AI agents?

AI agents are **smart software services that combine generative AI models with contextual data and the ability to automate tasks based on user input and environmental factors that they perceive**.

For example, an organization might build an AI agent to help employees manage expense claims. The agent might use a generative model combined with corporate expenses policy documentation to

answer employee questions about what expenses can be claimed and what limits apply. Additionally, the agent could use a programmatic function to automatically submit expense claims for regularly repeated expenses (such as a monthly cellphone bill) or intelligently route expenses to the appropriate approver based on claim amounts.

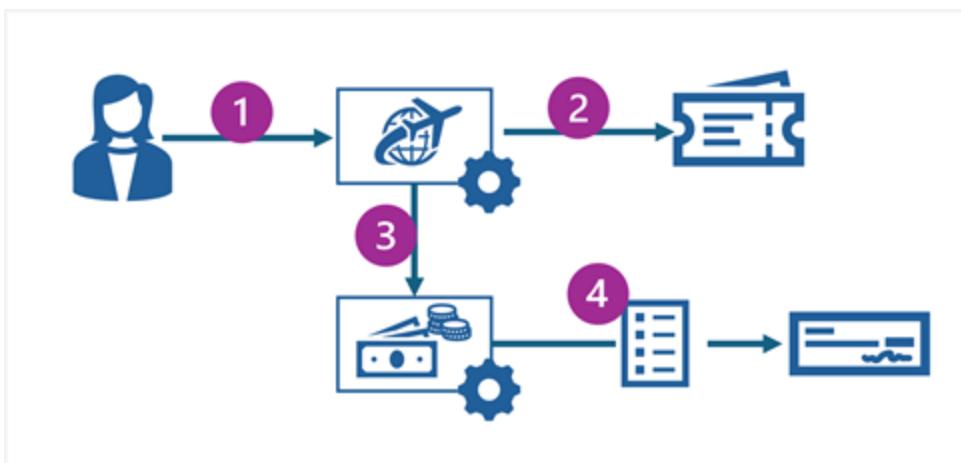
An example of the expenses agent scenario is shown in the following diagram.



The diagram shows the following process:

1. A user asks the expense agent a question about expenses that can be claimed.
2. The expenses agent accepts the question as a prompt.
3. The agent uses a knowledge store containing expenses policy information to ground the prompt.
4. The grounded prompt is submitted to the agent's language model to generate a response.
5. The agent generates an expense claim on behalf of the user and submits it to be processed and generate a check payment.

In more complex scenarios, organizations can develop **multi-agent solutions** in which multiple agents coordinate work between them. For example, a *travel booking agent* could book flights and hotels for employees and automatically submit expense claims with appropriate receipts to the expenses agent, as shown in this diagram:



The diagram shows the following process:

1. A user provides details of an upcoming trip to a travel booking agent.
2. The travel booking agent automates the booking of flight tickets and hotel reservations.
3. The travel booking agent initiates an expense claim for the travel costs through the expense agent.
4. The expense agent submits the expense claim for processing.

Options for agent development

There are many ways that developers can create AI agents, including **multiple frameworks and SDKs**.

Azure AI Foundry Agent Service

Azure AI Foundry Agent Service is a managed service in Azure that is designed to provide a framework for **creating, managing, and using AI agents within Azure AI Foundry**. The service is based on the OpenAI Assistants API but with increased choice of models, data integration, and enterprise security; enabling you to use both the **OpenAI SDK** and the **Azure Foundry SDK** to **develop agentic solutions**.

Tip: For more information about Foundry Agent Service, see the [Azure AI Foundry Agent Service documentation](#).

OpenAI Assistants API

The **OpenAI Assistants API** provides a subset of the features in Foundry Agent Service, and can only be used with OpenAI models. In Azure, you can use the Assistants API with Azure OpenAI, though in practice the Foundry Agent Service provides greater flexibility and functionality for agent development on Azure.

Semantic Kernel

Semantic Kernel is a **lightweight, open-source development kit** that you can use to build AI agents and orchestrate multi-agent solutions. The **core Semantic Kernel SDK** is designed for all kinds of generative AI development, while the **Semantic Kernel Agent Framework** is a platform specifically optimized for creating agents and implementing agentic solution patterns.

AutoGen

AutoGen is an open-source framework for developing agents rapidly. It's **useful as a research and ideation tool** when experimenting with agents.

Microsoft 365 Agents SDK

Developers can create self-hosted agents for delivery through a wide range of channels by using the **Microsoft 365 Agents SDK**. Despite the name, agents built using this SDK are not limited to Microsoft 365, but can be delivered through channels like **Slack** or **Messenger**.

Microsoft Copilot Studio

Microsoft Copilot Studio provides a **low-code development environment** that "citizen developers" can use to quickly build and deploy agents that integrate with a Microsoft 365 ecosystem or commonly used channels like Slack and Messenger. The visual design interface of Copilot Studio makes it a good choice for building agents when you have little or no professional software development experience.

Copilot Studio agent builder in Microsoft 365 Copilot

Business users can use the **declarative Copilot Studio agent builder tool** in Microsoft 365 Copilot to author basic agents for common tasks. The declarative nature of the tool enables users to create an agent by describing the functionality they need, or they can use an intuitive visual interface to specify options for their agent.

Choosing an agent development solution

With such a wide range of available tools and frameworks, it can be challenging to decide which ones to use. Use the following **considerations** to help you identify the right choices for your scenario:

- For business users with **little or no software development experience**, **Copilot Studio agent builder in Microsoft 365 Copilot Chat** provides a way to create simple declarative agents that automate everyday tasks. This approach can empower users across an organization to benefit from AI agents with minimal impact on IT.
- If business users have sufficient technical skills to build **low-code solutions using Microsoft Power Platform technologies**, Copilot Studio enables them to combine those skills with their business domain knowledge and build agent solutions that extend the capabilities of Microsoft 365 Copilot or add agentic functionality to common channels like Microsoft Teams, Slack, or Messenger.

- When an organization needs more complex extensions to Microsoft 365 Copilot capabilities, **professional developers** can use the **Microsoft 365 Agents SDK** to build agents that target the same channels as Copilot Studio.
- To develop agentic solutions that use Azure back-end services with a wide choice of models, custom storage and search services, and integration with Azure AI services, professional developers should use **Foundry Agent Service**.
- Start with **Foundry Agent Service** to develop single, standalone agents. When you need to build multi-agent solutions, use **Semantic Kernel** to orchestrate the agents in your solution.

User	Scenario	Tool
Little to no software dev experience	simple declarative agents that automate everyday work	Copilot Studio agent builder in Microsoft 365 Copilot Chat
Users with sufficient tech skills	Build low-code solutions using Microsoft Power Platform technologies	Microsoft 365 Copilot
Professional developers	Build agents that target the same channels as Copilot Studio	Microsoft 365 Agents SDK
Professional developers	Develop agentic solutions that use Azure back-end services with a wide choice of models, custom storage and search services	Foundry Agent Service
Professional developers	Develop single, standalone agents	Foundry Agent Service
Professional developers	multi-agent solutions	Semantic Kernel

Note: There's overlap between the capabilities of each agent development solution, and in some cases factors like existing familiarity with tools, programming language preferences, and other considerations will influence the decision.

Azure AI Foundry Agent Service

Azure AI Foundry Agent Service is a service within Azure that you can use to **create, test, and manage AI agents**. It provides both a visual agent development experience in the Azure AI Foundry

portal and **a code-first development experience using the Azure AI Foundry SDK**.

Components of an agent

Agents developed using Foundry Agent Service have the following elements:

- **Model**: A deployed generative AI model that enables the agent to reason and generate natural language responses to prompts. You can use common OpenAI models and a selection of models from the [Azure AI Foundry model catalog](#).
- **Knowledge**: data sources that enable the agent to ground prompts with contextual data. Potential knowledge sources include *Internet search results from Microsoft Bing, an Azure AI Search index, or your own data and documents*.
- **Tools**: Programmatic functions that enable the agent to automate actions. Built-in tools to access knowledge in Azure AI Search and Bing are provided as well as a code interpreter tool that you can use to generate and run Python code. You can also create custom tools using your own code or Azure Functions.

Conversations between users and agents take place on a thread, which retains a history of the messages exchanged in the conversation as well as any data assets, such as files, that are generated.

Exercise - Explore AI Agent development

In this exercise, you use the Azure AI Agent service in the Azure AI Foundry portal to create a simple AI agent that assists employees with expense claims.

Explore AI Agent development

In this exercise, you use the Azure AI Agent service in the Azure AI Foundry portal to create a simple AI agent that assists employees with expense claims.

Module assessment

1. Which of the following best describes an AI agent? **A software service that uses AI to assist users with information and task automation.**

2. Which AI agent development service offers a choice of generative AI models from multiple vendors in the Azure AI Foundry model catalog? **Azure AI Foundry Agent Service**.
3. What element of an Foundry Agent Service agent enables it to ground prompts with contextual data? **Knowledge**.

Summary

In this module, you learned about AI agents and some of the options available for developing them. You also learned how to create a simple agent using the visual tools for **Foundry Agent Service in the Azure AI Foundry portal**.

Tip: For more information about Foundry Agent Service, see [Azure AI Foundry Agent Service documentation](#).

Develop an AI agent with Azure AI Foundry Agent Service

This module provides engineers with the skills to begin **building agents with Azure AI Foundry Agent Service**.

Learning objectives

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

- Describe the **purpose of AI agents**
- Explain the **key features of Azure AI Foundry Agent Service**
- **Build an agent** using the Foundry Agent Service
- **Integrate an agent** in the Foundry Agent Service into your own application

Introduction

Azure AI Foundry Agent Service is a *fully managed service* designed to empower developers to securely **build, deploy, and scale high-quality, extensible AI agents** without needing to manage the underlying compute and storage resources.

Imagine you're working in the healthcare industry, where there's a need to automate patient interactions and streamline administrative tasks. Your organization wants to develop an AI agent that can handle patient inquiries, schedule appointments, and provide medical information based on real-time data. However, managing the infrastructure and ensuring data security are significant challenges. Azure AI Foundry Agent Service offers a solution by allowing you to create AI agents tailored to your needs through custom instructions and advanced tools. This service **simplifies the development process, reduces the amount of code required, and manages the underlying infrastructure, enabling you to focus on building high-quality AI solutions**.

In this module, you'll learn how to use the Foundry Agent Service to develop agents.

What is an AI agent

An AI agent is a software service that uses generative AI to understand and perform tasks on behalf of a user or another program. These agents use advanced AI models to understand context, make decisions, utilize grounding data, and take actions to achieve specific goals. Unlike traditional applications, AI agents can operate independently, executing complex workflows and automating processes without the need of constant human intervention. The evolution of generative AI enables agents to behave intelligently on our behalf, transforming how we can use and integrate these agents.

Understanding what an AI agent is and how to utilize them is crucial for effectively using AI to automate tasks, make informed decisions, and enhance user experiences. This knowledge enables organizations to deploy AI agents strategically, maximizing their potential to drive innovation, improve efficiency, and achieve business objectives.

Why Are AI agents useful?

AI agents are incredibly useful for several reasons:

- **Automation of Routine Tasks:** AI agents can handle repetitive and mundane tasks, freeing up human workers to focus on more strategic and creative activities. This leads to increased productivity and efficiency.
- **Enhanced Decision-Making:** By processing vast amounts of data and providing insights, AI agents support better decision-making. They can **analyze trends, predict outcomes, and offer recommendations based on real-time data**. AI Agents can even use advanced decision-making algorithms and machine learning models to analyze data and make informed decisions autonomously. This allows them to handle complex scenarios and provide actionable insights, whereas generative AI chat models primarily focus on generating text-based responses.
- **Scalability:** AI agents can scale operations without the need for proportional increases in human resources. This is beneficial for businesses looking to grow without significantly increasing operational costs.
- **24/7 Availability:** Like all software, AI agents can operate continuously without breaks, ensuring that tasks are completed promptly and customer service is available around the clock.

Agents are built to simulate human-like intelligence and can be applied in various domains such as customer service, data analysis, automation, and more.

Examples of AI agent use cases

AI agents have a wide range of applications across various industries. Here are some notable examples:

Personal productivity agents

Personal productivity agents assist individuals with **daily tasks** such as **scheduling meetings, sending emails, and managing to-do lists**. For instance, Microsoft 365 Copilot can help users draft documents, create presentations, and analyze data within the Microsoft Office suite.

Research agents

Research agents continuously **monitor market trends, gather data, and generate reports**. These agents can be used in financial services to **track stock performance**, in healthcare to stay updated with the latest **medical research**, or in marketing to analyze consumer behavior.

Sales agents

Sales agents automate lead generation and qualification processes. They can **research potential leads, send personalized follow-up messages, and even schedule sales calls**. This automation helps sales teams focus on closing deals rather than administrative tasks.

Customer service agents

Customer service agents **handle routine inquiries, provide information, and resolve common issues**. They can be integrated into chatbots on websites or messaging platforms, offering instant support to customers. For example, Cineplex uses an AI agent to process refund requests, significantly reducing handling time and improving customer satisfaction.

Developer agents

Developer agents help in **software development** tasks such as **code review, bug fixing, and repository management**. They can automatically **update codebases, suggest improvements**, and ensure that **coding standards are maintained**. GitHub Copilot is a great example of a developer agent.

Tip: To learn more about GitHub Copilot, explore the [GitHub Copilot fundamentals](#) learning path.

Note: You can explore more about agents in general with the [Fundamentals of AI agents](#) module.

How to use Azure AI Foundry Agent Service

Azure AI Foundry Agent Service is a fully managed service designed to empower developers to securely build, deploy, and scale high-quality, extensible AI agents without needing to manage the underlying compute and storage resources. This unit covers the purpose, benefits, key features, and integration capabilities of Azure AI Foundry Agent Service.

Purpose of Azure AI Foundry Agent Service

The Foundry Agent Service allows developers to **create AI agents tailored to their needs through custom instructions and advanced tools like code interpreters and custom functions**. These agents can answer questions, perform actions, or automate workflows by combining generative AI models with tools that interact with real-world data sources. The service simplifies the development process by reducing the amount of code required and managing the underlying infrastructure.

Previously, developers could create an agent-like experience by using standard APIs in Azure AI Foundry and connect to custom functions or other tools, but doing so would take a significant coding effort. Foundry Agent Service handles all of that for you through AI Foundry to build agents via the portal or in your own app in fewer than 50 lines of code. The exercise in the module explores both methods of building an agent.

Foundry Agent Service is ideal for **scenarios requiring advanced language models for workflow automation**. It can be used to:

- Answer questions using real-time or proprietary data sources.
- Make decisions and perform actions based on user inputs.
- Automate complex workflows by combining generative AI models with tools that interact with real-world data.

For example, an AI agent can be created to generate reports, analyze data, or even interact with users through apps or chatbots, making it suitable for customer support, data analysis, and automated reporting.

Key features of Foundry Agent Service

Foundry Agent Service offers several key features:

- **Automatic tool calling:** The service handles the entire tool-calling lifecycle, including running the model, invoking tools, and returning results.

- **Securely managed data:** Conversation states are securely managed using threads, eliminating the need for developers to handle this manually.
- **Out-of-the-box tools:** The service includes tools for file retrieval, code interpretation, and interaction with data sources like Bing, Azure AI Search, and Azure Functions.
- **Flexible model selection:** Developers can choose from various models, including Azure OpenAI models and others like Llama 3, Mistral, and Cohere.
- **Enterprise-grade security:** The service ensures data privacy and compliance with secure data handling and keyless authentication.
- **Customizable storage solutions:** Developers can use either platform-managed storage or bring their own Azure Blob storage for full visibility and control.

Foundry Agent Service provides a more streamlined and secure way to build and deploy AI agents compared to developing with the Inference API directly.

Foundry Agent Service resources

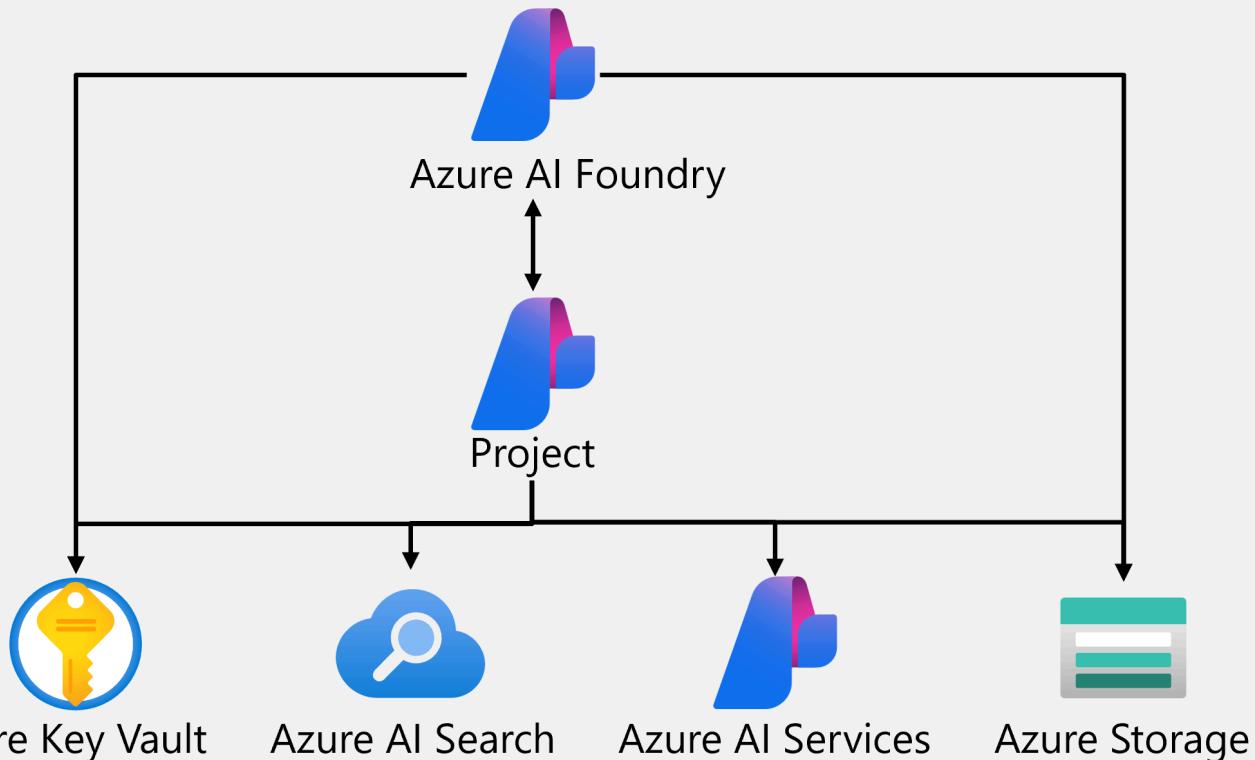
Foundry Agent Service is fully managed and designed to help developers build agents without having to worry about underlying resources. Through Azure, AI Foundry and the Agent Service will provision the necessary cloud resources. If desired, you can choose to connect your own resources when building your agent, giving you the flexibility to utilize Azure however works best for you.

At a minimum, you need to **create an Azure AI hub with an Azure AI project** for your agent. You can add more Azure services as required. You can **create the resources using the Azure AI Foundry portal**, or you can **use predefined bicep templates to deploy the resources** in your subscription.

Two common architectures for Foundry Agent Service solutions are:

- Basic agent setup: A minimal configuration that includes Azure AI hub, Azure AI project, and Azure AI Services resources.
- Standard agent setup: A more comprehensive configuration that includes the basic agent setup plus Azure Key Vault, Azure AI Search, and Azure Storage.

Standard agent setup



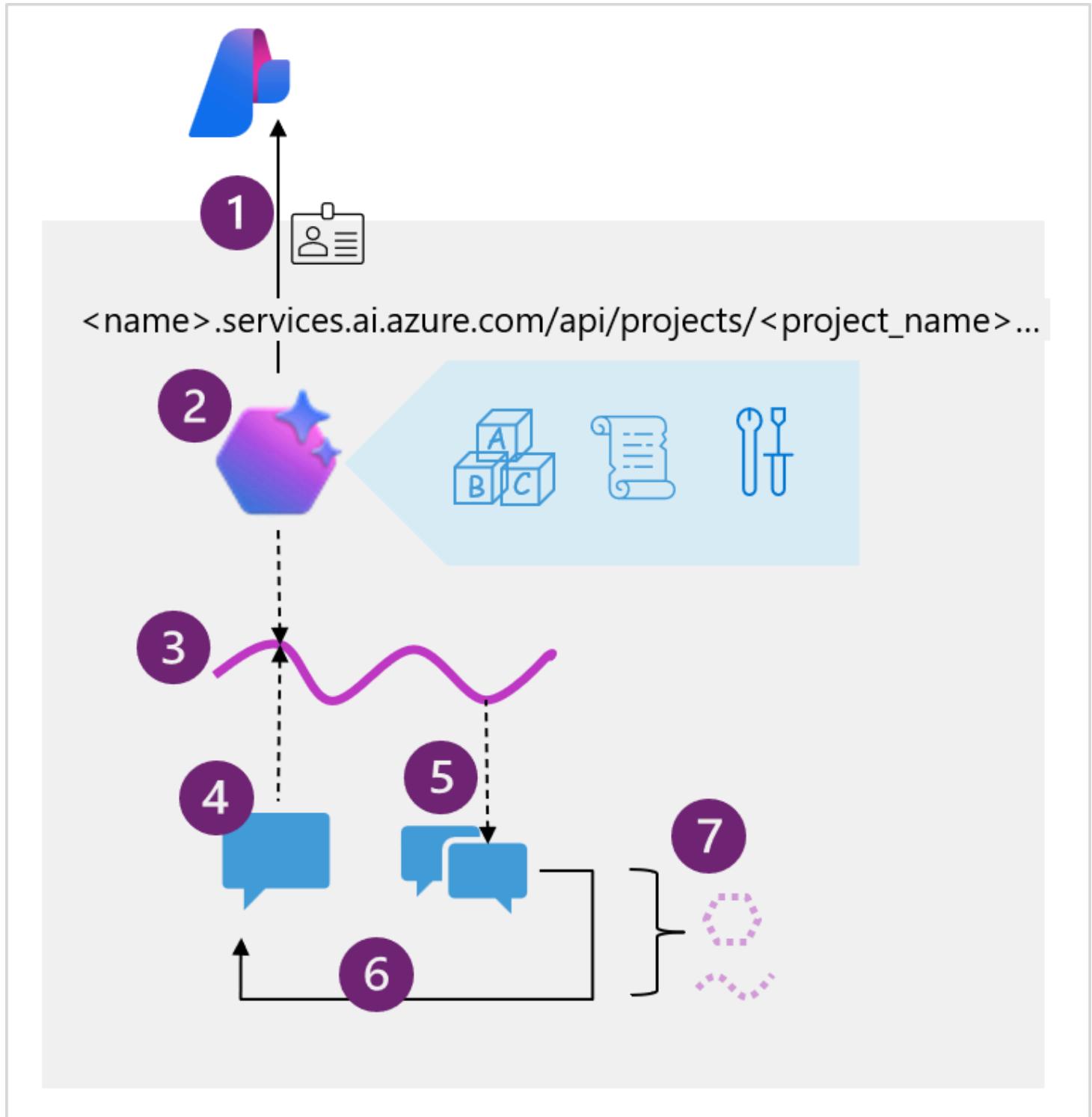
Develop agents with the Azure AI Foundry Agent Service

Previous solutions to achieve an agent-like experience took hundreds of lines of code to do things like referencing grounding data or connecting to a custom function. The Agent Service now simplifies all of that, supporting client-side function calling with just a few lines of code and connections to Azure Functions or an OpenAPI defined tool.

Note: Foundry Agent Service offers several advantages to building agents, but isn't always the right solution for your use case. For example, if you're trying to **build an integration with Microsoft 365** you might choose the **Copilot Studio agent builder** and if you're trying to **orchestrate multiple agents**, you might choose the **Semantic Kernel Agents Framework**. This [Fundamentals of AI Agents](#) unit explores more of the options for building agents.

Developing apps that use agents

Foundry Agent Service provides several **SDKs and a REST API** for you to integrate agents into your app using your preferred programming language. The exercise later in this module focuses on Python, but the overall pattern is the same for REST or other language SDKs.



The diagram shows the following high-level steps that you must implement in your code:

1. Connect to the *AI Foundry project* for your agent, using the project **endpoint and Entra ID authentication**.
2. Get a reference to an existing agent that you created in the Azure AI Foundry portal, or create a new one specifying:
 - The **model deployment** in the project that the agent should use to interpret and respond to prompts.
 - **Instructions** that determine the functionality and behavior of the agent.
 - **Tools and resources** that the agent can use to perform tasks.
3. Create a **thread** for a chat session with the agent. All conversations with an agent are conducted on a stateful thread that retains message history and data artifacts generated during the chat.
4. Add **messages** to the thread and invoke it with the agent.
5. Check the thread **status**, and when ready, retrieve the messages and data artifacts.
6. Repeat the previous two steps as a **chat loop** until the conversation can be concluded.
7. When finished, delete the agent and the thread to clean up the resources and delete data that is no longer required.

Tools available to your agent

Much of the enhanced functionality of an agent comes from the agent's ability to determine when and how to use **tools**. Tools make additional functionality available to your agent, and if the conversation or task warrants the use of one or more of the tools, the agent calls that tool and handle the response.

You can assign tools when creating an agent in the Azure AI Foundry portal, or when defining an agent in code using the SDK.

For example, one of the tools available is the **code interpreter**. This tool enables your agent to run custom code it writes to achieve something, such as MATLAB code to create a graph or solve a data analytics problem.

Available tools are split into two categories:

1. Knowledge tools

Knowledge tools enhance the context or knowledge of your agent. Available tools include:

- **Bing Search**: Uses Bing search results to ground prompts with real-time live data from the web.
- **File search**: Grounds prompts with data from files in a vector store.
- **Azure AI Search**: Grounds prompts with data from Azure AI Search query results.
- **Microsoft Fabric**: Uses the **Fabric Data Agent** to ground prompts with data from your Fabric data stores.

Tip: You can also integrate third-party licensed data by using the OpenAPI Spec action tool (discussed below).

2. Action tools

Action tools perform an action or run a function. Available tools include:

- **Code Interpreter:** A sandbox for model-generated Python code that can access and process uploaded files.
- **Custom function:** Call your custom function code – you must provide function definitions and implementations.
- **Azure Function:** Call code in serverless Azure Functions.
- **OpenAPI Spec:** Call external APIs based on the OpenAPI 3.0 spec.

By connecting built-in and custom tools, you can allow your agent to perform countless tasks on your behalf.

Exercise - Build an AI agent

Now it's your opportunity to build an agent in Azure AI Foundry. In this exercise, you create an agent and test it in Agent playground. You'll then develop your own app that integrates with an agent through Foundry Agent Service.

Develop an AI agent

In this exercise, you'll use Azure AI Agent Service to create a simple agent that analyzes data and creates charts. The agent uses the built-in Code Interpreter tool to dynamically generate the code required to create charts as images, and then saves the resulting chart images.

Module assessment

1. What is the first step in setting up Azure AI Foundry Agent Service? **Create an Azure AI Foundry Project.**
2. Which element of an agent definition is used to specify its behavior and restrictions? **Instructions.**

3. Which tool should you use to enable an agent to dynamically generate code to perform tasks or access data in files? **Code Interpreter**.

Summary

AI agents represent a significant advancement in the field of artificial intelligence, offering numerous benefits to businesses and individuals alike. By **automating routine tasks, enhancing decision-making, and providing scalable solutions**, AI agents are transforming how we work and interact with technology. As these agents continue to evolve, their potential applications will only expand, driving further innovation and efficiency across various sectors.

In this module, you learned about the purpose of **Foundry Agent Service**, its key features, the setup process, and its integration capabilities with other Azure AI services. We also addressed the challenge of building, deploying, and scaling AI agents. Foundry Agent Service solves several of these challenges, providing a fully managed environment for creating high-quality, extensible AI agents with minimal coding and infrastructure management.

The techniques covered in this module demonstrate several advantages, including **automatic tool calling, secure data management, and flexible model selection**. These features enable developers to focus on creating intelligent solutions while ensuring enterprise-grade security and compliance. The business impact includes streamlined development processes, reduced operational overhead, and enhanced AI capabilities.

After completing this module, you're now able to:

- Describe the **purpose of AI agents**
- Explain the **key features of Foundry Agent Service**
- **Build an agent** using the Foundry Agent Service
- **Integrate an agent** in the Foundry Agent Service into your own app

More reading:

- [Quickstart Guide for Azure AI Foundry Agent Service](#).
- [What's new in Azure AI Foundry Agent Service](#).
- For detailed instructions, see the [quickstart guide](#).

2.3 Develop AI agents with the Azure AI Foundry extension in Visual Studio Code

Learn how to build, test, and deploy AI agents using the **Azure AI Foundry extension in Visual Studio Code**.

Learning objectives

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

- Configure and deploy AI agents using the **agent designer**
- Add tools and capabilities to extend your agents' functionality
- Test agents using the integrated playground
- Generate sample code to integrate agents into applications

Introduction

As generative AI models become more powerful and accessible, developers are moving beyond simple chat applications to build intelligent agents that can automate complex tasks. These AI agents combine large language models with specialized tools to **access data, perform actions, and complete entire workflows** with minimal human intervention.

Visual Studio Code is a powerful platform for AI agent development, especially with the Azure AI Foundry extension. This extension brings enterprise-grade AI capabilities directly into your development environment. With the Azure AI Foundry for Visual Studio Code extension, you can discover and deploy models, create and configure agents, and test them in interactive playgrounds—all without leaving your code editor.

This module introduces the core concepts of AI agent development and shows you how to use the Azure AI Foundry extension for Visual Studio Code to build, test, and deploy intelligent agents. You'll learn how to use Azure AI Foundry's Agent Service to **create agents that can handle complex tasks, use various tools, and integrate with your existing applications**.

Get started with the Azure AI Foundry extension

The Azure AI Foundry for Visual Studio Code extension transforms your development environment into a comprehensive platform for building, testing, and deploying AI agents. This extension provides direct access to the capabilities of Azure AI Foundry's Agent Service without leaving your code editor, streamlining the entire agent development workflow.

What is the Azure AI Foundry for Visual Studio Code extension?

The Azure AI Foundry for Visual Studio Code extension is a powerful tool that brings enterprise-grade AI agent development capabilities directly into Visual Studio Code. It provides an integrated experience for:

- **Agent Discovery and Management** - Browse, create, and manage AI agents within your Azure AI Foundry projects
- **Visual Agent Designer** - Use an intuitive interface to configure agent instructions, tools, and capabilities
- **Integrated Testing** - Test agents in real-time using the built-in playground without switching contexts
- **Code Generation** - Generate sample integration code to connect agents with your applications
- **Deployment Pipeline** - Deploy agents directly to Azure AI Foundry for production use

Key features and capabilities

The extension offers several key features that accelerate AI agent development:

Agent Designer Interface

A visual designer that simplifies agent creation and configuration. You can define agent instructions, select appropriate models, and configure tools through an intuitive graphical interface.

Built-in Playground

An integrated testing environment where you can interact with your agents in real-time, test different scenarios, and refine agent behavior before deployment.

Tool Integration

Seamless integration with various tools including:

- **RAG (Retrieval-Augmented Generation)** for knowledge-based responses
- **Search** capabilities for information retrieval
- **Custom actions** for specific business logic
- **Model Context Protocol (MCP)** servers for extended functionality

Project Integration

Direct connection to your Azure AI Foundry projects, allowing you to work with existing resources and deploy new agents to your established infrastructure.

Installing the extension

To get started with the Azure AI Foundry for Visual Studio Code extension, first you need to have Visual Studio Code installed on your machine. You can download it from the [Visual Studio Code website](#).

You can install the Azure AI Foundry extension directly from the Visual Studio Code Marketplace:

1. Open Visual Studio Code.
2. Select Extensions from the left pane, or press Ctrl+Shift+X.
3. Search for and select Azure AI Foundry.
4. Select Install.
5. Verify the extension is installed successfully from the status messages.

Getting started workflow

The typical workflow for using the Azure AI Foundry extension follows these steps:

1. **Install and configure** the extension in Visual Studio Code
2. **Connect** to your Azure AI Foundry project
3. **Create or import** an AI agent using the designer
4. **Configure agent** instructions and add necessary tools
5. **Test the agent** using the integrated playground
6. **Iterate** on the design based on test results
7. **Generate code** for application integration

This streamlined workflow enables rapid prototyping and deployment of AI agents, making it easier for developers to build intelligent automation solutions that can handle complex real-world tasks.

The Azure AI Foundry for Visual Studio Code extension represents a significant step forward in making AI agent development more accessible and efficient, providing developers with enterprise-grade tools

in a familiar development environment.

Develop AI agents in Visual Studio Code

Creating and configuring AI agents in Visual Studio Code using the Azure AI Foundry extension provides a streamlined development experience that combines the power of Azure AI Foundry Agent Service with the familiar Visual Studio Code environment. This approach enables you to design, configure, and test agents without leaving your development environment.

Understanding Azure AI Foundry Agent Service

Azure AI Foundry Agent Service is **a managed service in Azure designed to provide a comprehensive framework for creating, managing, and deploying AI agents**. The service builds on the OpenAI Assistants API foundation while offering enhanced capabilities including:

- **Expanded model choice** - Support for multiple AI models beyond OpenAI
- **Enterprise security** - Built-in security features for production environments
- **Advanced data integration** - Seamless connection to Azure data services
- **Tooling ecosystem** - Access to a various built-in and custom tools

The Visual Studio Code extension provides direct access to these capabilities through an intuitive interface that simplifies the agent development process.

Creating agents with the extension

The Azure AI Foundry extension provides multiple ways to create AI agents, whether you're starting from scratch or building on existing work. The flexible approach accommodates different development preferences and scenarios.

Prerequisites for agent creation

Before creating an agent, complete the following steps:

1. Complete the extension setup and sign in to your Azure account
2. Create a default Azure AI Foundry project, or select an existing one
3. Select and deploy the model for your agent to use, or use an existing deployment

Creating a new agent

To create a new AI agent, follow these steps:

1. Open the Azure AI Foundry Extension view in Visual Studio Code
2. Navigate to the Resources section
3. Select the + (plus) icon next to the Agents subsection to create a new AI agent
4. Configure the agent properties in the Agent Designer view that opens

When you create an agent, the extension opens both the agent .yaml file and the Designer view, providing you with both a visual interface and direct access to the configuration file.

Configuring agent properties

Once you create an agent, the extension provides comprehensive configuration options to define how your agent behaves and interacts with users. The Agent Designer provides an intuitive interface for setting up these properties.

Basic Configuration

In the Agent Designer, configure the following essential properties:

- **Agent name** - Enter a descriptive name for your agent in the prompt
- **Model selection** - Choose your model deployment from the dropdown (this is the deployment name you chose when deploying a model)
- **Description** - Add a clear description of what your agent does
System instructions - Define the agent's behavior, personality, and response style
- **Agent ID** - Automatically generated by the extension

Understanding the agent YAML file

Your AI agent is defined in a YAML file that contains all configuration details. Here's an example structure:

```
yaml-language-server: $schema=https://aka.ms/ai-foundry-vsc/agent/1.0.0
version: 1.0.0
name: my-agent
description: Description of the agent
id: ''
metadata:
  authors:
    - author1
    - author2
  tags:
    - tag1
    - tag2
model:
  id: 'gpt-4o-1'
  options:
    temperature: 1
    top_p: 1
instructions: Instructions for the agent
tools: []
```

This YAML file is opened automatically alongside the Designer view, allowing you to work with either the visual interface or edit the configuration directly.

Agent instruction design

Well-crafted instructions are the foundation of effective AI agents. They define how your agent understands its role, responds to users, and handles various scenarios.

Best practices for instructions

When writing system instructions for your agent:

- **Be specific and clear** - Define exactly what the agent should do and how it should behave
- **Provide context** - Explain the agent's role and the environment it operates in
- **Set boundaries** - Clearly define what the agent should and shouldn't do
- **Include examples** - Show the agent examples of desired interactions when helpful
- **Define personality** - Establish the tone and style of responses

Instruction examples

For a customer service agent, effective instructions might include:

- The agent's role and purpose
- Guidelines for handling different types of customer inquiries
- Escalation procedures for complex issues
- Tone and communication style preferences

Deploying agents

Once you configure your agent, you can deploy it to Azure AI Foundry.

Deployment process

To deploy your agent:

- **Select the "Create on Azure AI Foundry" button** in the bottom-left of the Designer
- **Wait for deployment completion** - The extension handles the deployment process
- **Refresh the Azure Resources view** in the Visual Studio Code navbar
- **Verify deployment** - The deployed agent appears under the Agents subsection

Managing deployed agents

After deployment, you can:

- **View agent details** - Select the deployed agent to see the Agent Preferences page
- **Edit the agent** - Select "Edit Agent" to modify configuration and redeploy with the Update on Azure AI Foundry button
- **Generate integration code** - Select "Open Code File" to create sample code for using the agent
- **Test in playground** - Select "Open Playground" to interact with the deployed agent

Testing and iteration

The integrated playground enables immediate testing of your agent configuration, allowing you to validate behavior and make adjustments in real-time.

Using the playground

After configuring your agent, you can test it using the built-in playground:

- **Real-time conversations** - Chat with your agent to test responses
- **Instruction validation** - Verify the agent follows its configured instructions
- **Behavior testing** - Test how the agent handles different types of requests
- **Iterative refinement** - Make adjustments based on testing results

Working with agent threads

When you interact with deployed agents, **the system creates threads to manage conversation sessions:**

- **Threads** - Conversation sessions between an agent and user that store messages and handle context management
- **Messages** - Individual interactions that can include text, images, and files
- **Runs** - Single executions of an agent that use the agent's configuration and thread messages

You can view and manage these threads through the Azure Resources view in the extension.

Creating and configuring AI agents with the Azure AI Foundry Visual Studio Code extension provides a powerful yet accessible approach to agent development. The extension provides visual design tools, direct YAML editing, comprehensive configuration options, and integrated testing capabilities. These features enable developers to rapidly prototype and deploy sophisticated AI agents that can handle complex real-world scenarios.

Extend AI agent capabilities with tools

One of the most powerful features of AI agents is their ability to use tools that extend their capabilities beyond simple text generation. The Azure AI Foundry for Visual Studio Code extension makes it easy to add and configure tools for your agents. These tools enable agents to perform actions, access data, and integrate with external systems.

Understanding agent tools

Tools are programmatic functions that enable agents to automate actions and access information beyond their training data. When an agent determines that a tool is needed to respond to a user request, it can automatically invoke the appropriate tool, process the results, and incorporate them into its response. This capability transforms agents from simple text generators into powerful automation systems that can interact with real-world data and services.

Built-in tools

Azure AI Foundry provides several built-in tools that you can easily add to your agents without any additional configuration or setup. These tools are production-ready and handle common use cases that many agents require.

- **Code Interpreter** - Enables agents to **write and execute Python code for mathematical calculations, data analysis, chart generation, file processing, and complex problem-solving**
- **File Search** - Provides retrieval-augmented generation by uploading and indexing documents, searching knowledge bases, and supporting various file formats (PDF, Word, text files)
- **Grounding with Bing Search** - Allows agents to search the internet for real-time data, current events, and trending topics while providing citations and sources
- **OpenAPI Specified Tools** - Connects agents to external APIs and services through OpenAPI 3.0 specifications
- **Model Context Protocol (MCP)** - Standardized tool interfaces for extended functionality and community-driven tools

Adding tools in Visual Studio Code

The Azure AI Foundry extension provides an intuitive interface for adding tools to your agents through a streamlined process. The visual interface makes it easy to browse, configure, and test tools without writing any code:

- Select your agent in the extension
- Navigate to the Tools section in the configuration panel
- Browse available tools from the tool library
- Configure tool settings as needed
- Test tool integration using the playground

When you add a tool, you can also add any new assets it needs. For example, if you add a File Search tool, you can use an existing vector store asset or make a new asset for your vector store to host your uploaded files.

Model Context Protocol (MCP) servers

MCP servers provide a standardized way to add tools to your agents using an open protocol. This approach enables you to use community-built tools and create reusable components that work across different agent implementations.

Key benefits include:

- **Standardized protocol** for consistent tool communication
- **Reusable components** that work across different agents
- **Community-driven tools** available through MCP registries
- **Simplified integration** with consistent interfaces

The extension supports adding MCP servers through registry browsing, custom server addition, configuration management, and testing and validation.

Tool management and best practices

Effective tool management ensures your agents perform reliably and efficiently in production environments. Following best practices helps you avoid common pitfalls and optimize agent performance:

Tool Selection Guidelines

- Identify what capabilities your agent requires
- Start with built-in tools before adding custom solutions
- Test thoroughly to validate tool behavior in various scenarios
- Monitor performance to track tool usage and effectiveness

Adding tools and extending agent capabilities through the Azure AI Foundry Visual Studio Code extension enables you to create sophisticated AI agents that can handle complex real-world tasks. By combining built-in tools with custom functions and MCP servers, you can build agents that seamlessly integrate with your existing systems and business processes while maintaining enterprise-grade security and performance.

Exercise - Build an AI agent using the Azure AI Foundry extension

If you have an Azure subscription, you can explore the Azure AI Foundry Extension for Visual Studio Code by completing this hands-on exercise. This exercise guides you through creating and deploying an AI agent using the extension.

Develop an AI agent with VS Code extension

In this exercise, you'll use the Azure AI Foundry VS Code extension to create an agent that can use Model Context Protocol (MCP) server tools to access external data sources and APIs. The agent will be able to retrieve up-to-date information and interact with various services through MCP tools.

Module assessment

1. When you create a new agent in the Azure AI Foundry extension, what two views are automatically opened? The YAML file and the Designer view
2. What is a key benefit of using Model Context Protocol (MCP) servers for agent tools? They provide reusable components that work across different agents
3. How does the Azure AI Foundry Agent Service manage conversation sessions when users interact with deployed agents? The system creates individual threads for each conversation to manage context and message history.

Summary

In this module, you learned how to develop AI agents using the Azure AI Foundry for Visual Studio Code extension. You explored how to create, configure, and test agents directly within your development environment. You also learned to extend their capabilities with various tools and deploy them to production environments.

Tip: For more information, see [Work with the Azure AI Foundry for Visual Studio Code extension](#).

2.4 Integrate custom tools into your agent

Built-in tools are useful, but they may not meet all your needs. In this module, learn how to extend the capabilities of your agent by **integrating custom tools** for your agent to use.

Learning objectives

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

- Describe the benefits of **using custom tools** with your agent.
- Explore the different options for custom tools.
- **Build an agent that integrates custom tools** using the **Azure AI Foundry Agent Service**.

Introduction

Azure AI Foundry Agent Service offers a seamless way to build an agent without needing extensive AI or machine learning expertise. By using tools, you can provide your agent with functionality to execute actions on your behalf.

The AI Agent Service provides built-in tools for gathering knowledge and generating code, which provide your agent with some powerful functionality. However, sometimes your agent needs to be able to complete specific tasks or actions that an AI model would struggle to handle on its own. To accomplish these actions, you can **provide your agent a custom tool based on your own code or a third-party service or API**.

Imagine you're working in the retail industry, and your company is struggling with managing customer inquiries efficiently. The customer support team is overwhelmed with repetitive questions, leading to delays in response times and decreased customer satisfaction. By using Foundry Agent Service with custom tools, you can **create a custom FAQ agent that handles common inquiries**. This agent can be provided with a set of custom tools to look up customer orders, freeing up your support team to focus on more complex issues.

In this module, you'll learn how to **utilize custom tools in Foundry Agent Service** to enhance productivity, improve accuracy, and create tailored solutions for specific needs.

Why use custom tools

Azure AI Foundry Agent Service offers a powerful platform for integrating custom tools to enhance productivity and provide tailored solutions for specific business needs. By using these custom tools, businesses can achieve greater efficiency and effectiveness in their operations.

Why use custom tools?

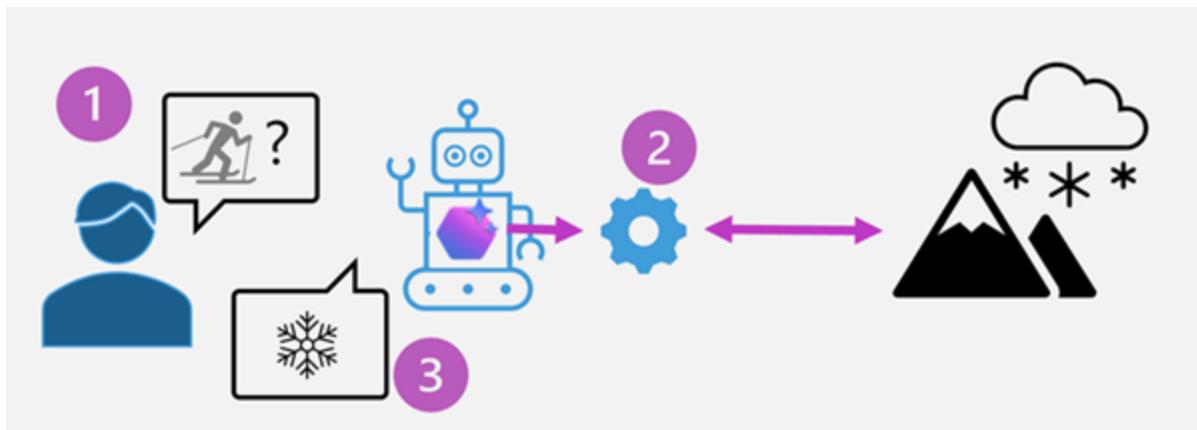
Custom tools in Azure AI services can significantly **enhance productivity** by automating repetitive tasks and streamlining workflows that are specific to your use case. These tools improve accuracy by providing precise and consistent outputs, reducing the likelihood of human error. Additionally, custom tools offer tailored solutions that address specific business needs, enabling organizations to optimize their processes and achieve better outcomes.

- **Enhanced productivity:** Automate repetitive tasks and streamline workflows.
- **Improved accuracy:** Provide precise and consistent outputs, reducing human error.
- **Tailored solutions:** Address specific business needs and optimize processes.

Adding tools makes custom functionality available for the agent to use, depending on how it decides to respond to the user prompt. For example, consider how **a custom tool to retrieve weather data from an external meteorological service** could be used by an agent.

The diagram shows the process of an agent choosing to use the custom tool:

1. A user asks an agent about the weather conditions in a ski resort.
2. The agent determines that it has access to a tool that can use an API to get meteorological information, and calls it.
3. The tool returns the weather report, and the agent informs the user.



Common scenarios for custom tools in agents

Custom tools within the Foundry Agent Service enable users to extend the capabilities of AI agents, tailoring them to meet specific business needs. Some example use cases that illustrate the versatility and impact of custom tools include:

Customer support automation

- **Scenario:** A retail company integrates a custom tool that connects the Azure AI Agent to their **customer relationship management (CRM) system**.
- **Functionality:** The AI agent can **retrieve customer order histories, process refunds, and provide real-time updates on shipping statuses**.
- **Outcome:** Faster resolution of customer queries, reduced workload for support teams, and improved customer satisfaction.

Inventory management

- **Scenario:** A manufacturing company develops a custom tool to link the AI agent with their **inventory management system**.
- **Functionality:** The AI agent can **check stock levels, predict restocking needs using historical data, and place orders with suppliers automatically**.
- **Outcome:** Streamlined inventory processes and optimized supply chain operations.

Healthcare appointment scheduling

- **Scenario:** A healthcare provider integrates a **custom scheduling tool** with the AI agent.
- **Functionality:** The AI agent can **access patient records, suggest available appointment slots, and send reminders to patients**.
- **Outcome:** Reduced administrative burden, improved patient experience, and better resource utilization.

IT Helpdesk support

- **Scenario:** An IT department develops a custom tool to integrate the AI agent with their **ticketing and knowledge base systems**.
- **Functionality:** The AI agent can **troubleshoot common technical issues, escalate complex problems, and track ticket statuses**.
- **Outcome:** Faster issue resolution, reduced downtime, and improved employee productivity.

E-learning and training

- **Scenario:** An educational institution creates a custom tool to connect the AI agent with their **learning management system (LMS)**.
- **Functionality:** The AI agent can **recommend courses, track student progress, and answer questions about course content**.
- **Outcome:** Enhanced learning experiences, increased student engagement, and streamlined administrative tasks.

These examples demonstrate how **custom tools within the Foundry Agent Service** can be used across industries to address unique challenges, drive efficiency, and deliver value.

Options for implementing custom tools

Azure AI Foundry Agent Service offers various custom tools that enhance the capabilities and efficiency of your AI agents. These tools allow for scalable interoperability with various applications, making it easier to integrate with existing infrastructure or web services.

Custom tool options available in Azure AI Foundry Agent Service

Azure AI services provide several custom tool options, including **OpenAPI specified tools, Azure Functions, and function calling**. These tools enable seamless integration with **external APIs, event-driven applications, and custom functions**.

- **Custom function:** Function calling allows you to *describe the structure of custom functions to an agent and return the functions that need to be called along with their arguments*. The agent can dynamically identify appropriate functions based on their definitions. This feature is useful for integrating custom logic and workflows, in a selection of programming languages, into your AI agents.
- **Azure Functions:** Azure Functions enable you to create intelligent, event-driven applications with minimal overhead. They support **triggers and bindings**, which simplify how your AI Agents interact with external systems and services. **Triggers determine when a function executes, while bindings facilitate streamlined connections to input or output data sources**.
- **OpenAPI specification tools:** These tools allow you to connect your Azure AI Agent to an external API using an **OpenAPI 3.0 specification**. This provides standardized, automated, and scalable API integrations that enhance the capabilities of your agent. OpenAPI specifications

describe HTTP APIs, enabling people to understand how an API works, generate client code, create tests, and apply design standards.

- **Azure Logic Apps:** This action provides **low-code/no-code solutions** to add workflows and connects apps, data, and services with the **low-code Logic App**.

This flexibility to integrate custom functionality in multiple ways enables a wide range of extensibility possibilities for your Foundry Agent Service agents.

How to integrate custom tools

Custom tools in an agent can be defined in a handful of ways, depending on what works best for your scenario. You may find that your company already has **Azure Functions** implemented for your agent to use, or a **public OpenAPI specification** gives your agent the functionality you're looking for.

Function Calling

Function calling allows agents to execute predefined functions dynamically based on user input. This feature is ideal for scenarios where agents need to perform specific tasks, such as retrieving data or processing user queries, and can be done in code from within the agent. Your function may call out to other APIs to get additional information or initiate a program.

Example: Defining and using a function

Start by defining a function that the agent can call. For instance, here's a fake snowfall tracking function:

```

import json

def recent_snowfall(location: str) -> str:
    """
    Fetches recent snowfall totals for a given location.

    :param location: The city name.
    :return: Snowfall details as a JSON string.
    """

    mock_snow_data = {"Seattle": "0 inches", "Denver": "2 inches"}
    snow = mock_snow_data.get(location, "Data not available.")
    return json.dumps({"location": location, "snowfall": snow})

user_functions: Set[Callable[..., Any]] = {
    recent_snowfall,
}

```

Register the function with your agent using the Azure AI SDK:

```

# Initialize agent toolset with user functions
functions = FunctionTool(user_functions)
toolset = ToolSet()
toolset.add(functions)
agent_client.enable_auto_function_calls(toolset=toolset)

# Create your agent with the toolset
agent = agent_client.create_agent(
    model="gpt-4o-mini",
    name="snowfall-agent",
    instructions="You are a weather assistant tracking snowfall. Use the provided functions to :",
    toolset=toolset
)

```

The agent can now call `recent_snowfall` dynamically when it determines that the prompt requires information that can be retrieved by the function.

Azure Functions

Azure Functions provide **serverless computing capabilities for real-time processing**. This integration is ideal for event-driven workflows, **enabling agents to respond to triggers such as HTTP requests or queue messages**.

Example: Using Azure Functions with a queue trigger

First, develop and deploy your Azure Function. In this example, imagine we have a function in our Azure subscription to fetch the snowfall for a given location.

When your Azure Function is in place, integrate add it to the agent definition as an Azure Function tool:

```
storage_service_endpoint = "https://<your-storage>.queue.core.windows.net"

azure_function_tool = AzureFunctionTool(
    name="get_snowfall",
    description="Get snowfall information using Azure Function",
    parameters={
        "type": "object",
        "properties": {
            "location": {"type": "string", "description": "The location to check snowfall."},
        },
        "required": ["location"],
    },
    input_queue=AzureFunctionStorageQueue(
        queue_name="input",
        storage_service_endpoint=storage_service_endpoint,
    ),
    output_queue=AzureFunctionStorageQueue(
        queue_name="output",
        storage_service_endpoint=storage_service_endpoint,
    ),
)

agent = agent_client.create_agent(
    model=os.environ["MODEL_DEPLOYMENT_NAME"],
    name="azure-function-agent",
    instructions="You are a snowfall tracking agent. Use the provided Azure Function to fetch sr
    tools=azure_function_tooldefinitions,
)
```

The agent can now send requests to the Azure Function via a storage queue and process the results.

OpenAPI Specification

OpenAPI defined tools allow agents to interact with external APIs using standardized specifications. This approach simplifies API integration and ensures compatibility with various

services. The Foundry Agent Service uses **OpenAPI 3.0** specified tools.

Tip: Currently, three authentication types are supported with OpenAPI 3.0 tools: **anonymous**, **API key**, and **managed identity**.

Example: Using an OpenAPI specification

First, create a JSON file (in this example, called `snowfall_openapi.json`) describing the API.

Then, register the OpenAPI tool in the agent definition:

```
from azure.ai.agents.models import OpenApiTool, OpenApiAnonymousAuthDetails

with open("snowfall_openapi.json", "r") as f:
    openapi_spec = json.load(f)

    auth = OpenApiAnonymousAuthDetails()
    openapi_tool = OpenApiTool(name="snowfall_api", spec=openapi_spec, auth=auth)

    agent = agent_client.create_agent(
        model="gpt-4o-mini",
        name="openapi-agent",
        instructions="You are a snowfall tracking assistant. Use the API to fetch snowfall data.",
        tools=[openapi_tool]
    )
```

The agent can now use the OpenAPI tool to fetch snowfall data dynamically.

Note: One of the concepts related to agents and custom tools that developers often have difficulty with is the declarative nature of the solution. **You don't need to write code that explicitly calls your custom tool functions - the agent itself decides to call tool functions based on messages in prompts.** By providing the agent with functions that have meaningful names and well-documented parameters, the agent can "figure out" when and how to call the function all by itself!

By using one of the available custom tool options (or any combination of them), you can create powerful, flexible, and intelligent agents with Foundry Agent Service. These integrations enable seamless interaction with external systems, real-time processing, and scalable workflows, making it easier to build custom solutions tailored to your needs.

Exercise - Build an agent with custom tools

Now it's your opportunity to build an agent with custom tools. In this exercise, you create an agent in code and connect the tool definition to a custom tool function.

Use a custom function in an AI agent

In this exercise you'll explore creating an agent that can use custom functions as a tool to complete tasks.

You'll build a simple technical support agent that can collect details of a technical problem and generate a support ticket.

Module assessment

1. What are custom tools, and how can they help you develop effective agents with Azure AI Foundry Agent Service? **Callable functions that an agent can use to extend its capabilities.**
2. You need to integrate functionality from an OpenAPI 3.0-based web service into an agent solution. What should you do? **Add the web services as an OpenAPI specification tool to the agent definition.**
3. Your agent application code includes a local function that you want the agent to call. What kind of tool should you add to the agent's definition? **Function calling.**

Summary

In this module, we covered the benefits of **integrating custom tools into Foundry Agent Service** to boost productivity and provide tailored business solutions. By providing custom tools to our agent, we can optimize processes to meet specific needs, resulting in better responses from your agent.

The techniques learned in this module enable businesses to generate marketing materials, improve communications, and analyze market trends more effectively, all through custom tools. The integration of various tool options in the AI Agent Service, from Azure Functions to OpenAPI specifications, allows for the creation of intelligent, event-driven applications that use well-established patterns already used in many businesses.

Further reading

- [AI Agents for beginners tool use](#)
- [Azure AI Foundry Agent Service function calling](#)
- [Introduction to Azure Functions](#)
- [OpenAPI Specification](#)

Develop a multi-agent solution with Azure AI Foundry Agent Service

Break down complex tasks with intelligent collaboration. Learn how to design multi-agent solutions using **connected agents**.

Learning objectives

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

- Describe how **connected agents** enable modular, collaborative workflows.
- Design a **multi-agent solution** by defining **main agent tools** and **connected agent roles**.
- Build and run a connected agent solution

Introduction

Azure AI Foundry Agent Service can help you develop sophisticated, **multi-agent systems** that can break down complex tasks into smaller, specialized roles. Using **connected agents**, you can design intelligent solutions where a primary agent delegates work to sub-agents without the need for custom orchestration logic or hardcoded routing. This modular approach boosts efficiency and maintainability across a wide range of scenarios.

Imagine you're on an engineering team that receives a constant flow of support tickets including bugs, feature requests, and infrastructure issues. Manually reviewing and sorting each one takes time and slows down your team's ability to respond quickly. With a **multi-agent approach**, you can build a triage assistant that assigns different tasks to specialized agents. The sub-agents might perform tasks such as classifying ticket type, setting priority, and suggesting the right team for the work. With a multi-agent approach, these specialized agents can work together to streamline the ticketing process.

In this module, you learn how to use connected agents in Azure AI Foundry. You also practice building **an intelligent ticket triage system using a collaborative multi-agent solution**.

Understand connected agents

As AI solutions become more advanced, managing complex workflows gets harder. A single agent can handle a wide range of tasks, but this approach can become unmanageable as the scope expands. That's why Azure AI Foundry Agent Service lets you connect multiple agents, each with a focused role, to work together in a cohesive system.

What are connected agents?

Connected agents are a feature in the Azure AI Foundry Agent Service that allows you to break large tasks into smaller, specialized roles without building a custom orchestrator or hardcoding routing logic. Instead of relying on one agent to do everything, you can create multiple agents with clearly defined responsibilities that collaborate to accomplish tasks.

At the center of this system, there's **a main agent that interprets user input and delegates tasks to connected sub-agents**. Each sub-agent is designed to perform a specific function, such as to summarize a document, validate a policy, or retrieve data from a knowledge source.

This division of labor helps you:

- Simplify complex workflows
- Improve agent performance and accuracy
- Make systems easier to maintain and extend over time

Why use connected agents?

Rather than scaling a single agent to handle every user request or data interaction, using connected agents lets you:

- Build modular solutions that are easier to develop and debug
- Assign specialized capabilities to agents that can be reused across solutions
- Scale your system in a way that aligns with real-world business logic

This approach is especially useful in scenarios where agents need to perform sensitive tasks independently, such as handling private data or generating personalized content.

Using connected agents to automate workflows offers many benefits, for example:

- **No custom orchestration required** - The main agent uses natural language to route tasks, eliminating the need for hardcoded logic.

- **Improved reliability and traceability** - The clear separation of responsibilities makes issues easier to debug since agents can be tested individually.
- **Flexible and extensible** - Add or swap agents without reworking the entire system or modifying the main agent.

Connected agents make it easier to build modular, collaborative systems without complex orchestration. By assigning focused roles and using natural language delegation, you can simplify workflows, improve reliability, and scale your solutions more effectively.

Design a multi-agent solution with connected agents

In a connected agent solution, **success depends on clearly defining the responsibilities of each agent. The central agent is also responsible for how the agents will collaborate**. Let's explore how to design a multi-agent program using Azure AI Foundry Agent Service.

Main agent (orchestrator) responsibilities

The main agent acts as the orchestrator. It *interprets the intent behind a request and determines which connected agent is best suited to handle it*. The main agent is responsible for:

- Interpreting user input
- Selecting the appropriate connected agent
- Forwarding relevant context and instructions
- Aggregating or summarize results

Connected agent responsibilities

Connected agents designed to focus on **a single domain of responsibility**. A connected agent is responsible for:

- Completing a specific action based on a clear prompt
- Using tools (if needed) to complete their task
- Returning the results to the main agent

Connected agents should be designed with a **single responsibility** in mind. This makes your system easier to debug, extend, and reuse.

Set up a multi-agent solution with connected agents

1. Initialize the agents client

First, you create a client that connects to your Azure AI Foundry project.

2. Create an agent to connect to the main agent

Define an agent you want to connect to the main agent. You can do this using the `create_agent` method on the `AgentsClient` object.

For example, your connected agent might retrieve stock prices, summarize documents, or validate compliance. Give the agent clear instructions that define its purpose.

3. Initialize the connected agent tool

Use your agent definition to create a `ConnectedAgentTool`. Assign it a name and description so the main agent knows when and how to use it.

4. Create the main agent

Create the main agent using the `create_agent` method. Add your connected agents using the `tools` property and assign the `ConnectedAgentTool` definitions to the main agent.

5. Create a thread and send a message

Create the **agent thread** that is used to manage the conversation context. Then create a message on the thread that contains the request you want the agent to fulfill.

6. Run the agent workflow

Once the message is added, create a run to process the request. The main agent uses its tools to delegate tasks as needed and compile a final response for the user.

7. Handle the results

When the run completes, you can review the main agent's response. The final output may incorporate insights from one or more connected agents. Only the main agent's response is visible to the end user.

Designing a connected agent system involves **defining focused agents, registering them as tools, and configuring a main agent to route tasks** intelligently. This modular approach gives you a flexible foundation for building collaborative AI solution that scale as your needs grow.

Exercise - Develop a multi-agent app with Azure AI Foundry

Develop a multi-agent solution

In this exercise, you'll create a project that orchestrates multiple AI agents using **Azure AI Foundry Agent Service**. You'll design an AI solution that assists with ticket triage. The connected agents will assess the ticket's priority, suggest a team assignment, and determine the level of effort required to complete the ticket. Let's get started!

Module assessment

1. What is the role of the main agent in a connected agent system? **To coordinate user input and route tasks to the appropriate connected agents.**
2. How do you connect an agent to a main agent using the Azure AI Projects client library? **Add the agent as a `ConnectedAgentTool` to the main agent's tool definition.**
3. How does the main agent decide which connected agent to use? **It uses prompt instructions and natural language understanding.**

Summary

In this module, you learned how to **design and implement multi-agent solutions** using **Azure AI Foundry Agent Service**.

Connected agents let you break down complex tasks by assigning them to specialized agents that work together within a coordinated system. You explored how to define clear roles for main and connected agents, delegate tasks using natural language, and design modular workflows that are easier to scale and maintain. You also practiced building a multi-agent solution. Great work!

Integrate MCP Tools with Azure AI Agents

Enable dynamic tool access for your Azure AI agents. Learn how to **connect MCP-hosted tools and integrate them seamlessly into agent workflows**.

Learning objectives

After completing this module, you're able to:

- Explain the roles of the **MCP server and client** in tool discovery and invocation.
- Wrap **MCP tools** as asynchronous functions and **register them with Azure AI agents**.
- Build an Azure AI agent that dynamically accesses and calls MCP tools during runtime.

Introduction

AI agents are capable of performing a wide range of tasks, but many tasks still require them to **interact with tools outside the large language model**. Agents may need to access APIs, databases, or internal services. Manually integrating and maintaining these tools can quickly become complex, especially as your system grows, or changes frequently.

Model Context Protocol (MCP) servers can help solve this problem by integrating with AI agents. Connecting an Azure AI Agent to a Model Context Protocol (MCP) server can provide your agent with a catalog of tools accessible on demand. This approach makes your AI solution more robust, scalable, and easier to maintain.

Suppose you're working for a retailer that specializes in cosmetics. Your team wants to build an AI assistant that can help manage inventory by checking product stock levels and recent sales trends. Using an MCP server, you can connect the assistant to a set of tools that can make inventory assessments and provide recommendations to the team.

In this module, you learn how to **set up an MCP server and client**, and connect tools to an Azure AI Agent dynamically. You also practice **creating your own AI MCP tool solution** with Azure AI Foundry Agent Service.

Understand MCP tool discovery

As AI agents become more capable, the range of tools and services they can access also grows. However, registering new tools, managing, updating, and integrating them can quickly become complex and time-consuming. **Dynamic tool discovery** helps solve this problem by enabling agents to find and use tools automatically at runtime.

What is *dynamic tool discovery*?

Dynamic tool discovery is a mechanism that allows an AI agent to discover available external tools without needing hardcoded knowledge of each one. Instead of manually adding or updating every tool your agent can use, the agent queries a centralized Model Context Protocol (MCP) server. This server acts as a live catalog, exposing tools that the agent can understand and call.

This approach means:

- Tools can be added, updated, or removed centrally without modifying the agent code.
- Agents can always use the latest version of a tool, improving accuracy and reliability.
- The complexity of managing tools shifts away from the agent and into a dedicated service.

How does MCP enable dynamic tool discovery?

An MCP server hosts a set of functions that are exposed as tools using the `@mcp.tool` decorator.

Tools are a primitive type in the MCP that enables servers to expose executable functionality to clients. A client can connect to the server and fetch these tools dynamically. The client then generates function wrappers that are added to the Azure AI Agent's tool definitions. This setup creates a flexible pipeline:

- The **MCP server hosts available tools**.
- The **MCP client dynamically discovers the tools**.
- The **Azure AI Agent uses the available tools** to respond to user requests.

Why use dynamic tool discovery with MCP?

This approach provides several benefits:

- **Scalability:** Easily add new tools or update existing ones without redeploying agents.
- **Modularity:** Agents can remain simple, focusing on delegation rather than managing tool details.
- **Maintainability:** Centralized tool management reduces duplication and errors.
- **Flexibility:** Supports diverse tool types and complex workflows by aggregating capabilities.

Dynamic tool discovery is especially useful in environments where tools evolve rapidly or where many teams manage different APIs and services. Using tools allows AI agents to adapt to changing capabilities in real time, interact with external systems securely, and perform actions that go beyond language generation.

Integrate agent tools using an MCP server and client

To dynamically connect tools to your Azure AI Agent, you first need a functioning Model Context Protocol (MCP) setup. This includes both the **MCP server, which hosts your tool catalog**, and the **MCP client, which fetches those tools and makes them usable by your agent**.

What is the MCP Server?

The **MCP server acts as a registry for tools** your agent can use. You can initialize your MCP server using `FastMCP("server-name")`. The FastMCP class uses Python type hints and document strings to automatically generate tool definitions, making it easy to create and maintain MCP tools. These definitions are then served over HTTP when requested by the client. Because tool definitions live on the server, you can update or add new tools at any time, without having to modify or redeploy your agent.

What is the MCP Client?

A standard **MCP client acts as a bridge between your MCP server and the Azure AI Agent Service**. The client initializes an MCP client session and connects to the server. Afterwards, it performs three key tasks:

- **Discovers available tools from the MCP server** using `session.list_tools()`.
- **Generates Python function stubs** that wrap the tools.
- **Registers those functions with your agent**.

This allows the agent to call any tool listed in the MCP catalog as if it were a native function, all without hardcoded logic.

Register tools with an Azure AI Agent

When an MCP client session is initialized, the client can dynamically pull in tools from the MCP server. An MCP tool can be invoked using `session.call_tool(tool_name, tool_args)`. **The tools should each be wrapped in an async function so that the agent is able to invoke them**. Finally, those

functions are bundled together and become part of the agent's toolset and are available during runtime for any user request.

Overview of MCP agent tool integration

- The **MCP server** hosts **tool definitions** decorated with `@mcp.tool`.
- The **MCP client** initializes an MCP client connection to the server.
- The **MCP client fetches the available tool definitions** with `session.list_tools()`.
- **Each tool is wrapped in an async function** that invokes `session.call_tool`
- The tool functions are bundled into `FunctionTool` that makes them usable by the agent.
- The `FunctionTool` is registered to the agent's toolset.

Now your agent is able to access and invoke your tools through natural language interaction. By setting up the MCP server and client, you create a clean separation between tool management and agent logic—enabling your system to adapt quickly as new tools become available.

Use Azure AI agents with MCP servers

You can **enhance your Azure AI Foundry agent by connecting it to Model Context Protocol (MCP) servers**. MCP servers provide tools and contextual data that your agent can use to perform tasks, extending its capabilities beyond built-in functions. Azure AI Agent Service includes support for remote MCP servers, allowing your agent to quickly connect to your server and access tools.

When you use the Azure AI Foundry Agent Service to connect to your MCP server, you don't need to manually create an MCP client session or add any function tools to your agent. Instead, you **create an MCP tool object that connects to your MCP server**. Then you add information about the MCP server to the agent thread when invoking a prompt. This also allows you to connect and use different tools from multiple servers depending on your needs.

Integrating remote MCP servers

To connect to an MCP server, you need:

- A remote MCP server endpoint ([example](#)).
- An Azure AI Foundry agent configured to use the MCP tool.

You can connect to multiple MCP servers by adding them as separate tools, each with:

- `server_label` : A unique identifier for the MCP server (e.g., GitHub).

- `server_url` : The MCP server's URL.
- `allowed_tools` (*optional*): A list of specific tools the agent is allowed to access.

The MCP tool also supports custom headers, which let you pass:

- Authentication keys (e.g., API keys, OAuth tokens).
- Other required headers for the MCP server.
 - These headers are included in `tool_resources` during each run and are not stored between runs.

Invoking tools

When using the Azure MCP Tool object, you don't need to wrap function tools or invoke `session.call_tool`. Instead, the tools are automatically invoked when necessary during an agent run.

To automatically invoke MCP tools:

- Create the `McpTool` object with the server label and URL.
- Use `update_headers` to apply any headers required by the server.
- Use the `set_approval_mode` to determine whether approval is required. Supported values are:
 - `always` : A developer needs to provide approval for every call. If you don't provide a value, this one is the default.
 - `never` : No approval is required.
- Create a `ToolSet` object and add the `McpTool` object
- Create an agent run and specify the `toolset` property
- When the run completes, you should see the results of any invoked tools in the response.

If the model tries to invoke a tool in your MCP server with approval required, you get a run status of `requires_action`. - In the `requires_action` field, you can get more details on which tool in the MCP server is called and any arguments to be passed. - Review the tool and arguments so that you can make an informed decision for approval. - Submit your approval to the agent with `call_id` by setting `approve` to true.

MCP integration is a key step toward creating richer, more context-aware AI agents. As the MCP ecosystem grows, you'll have even more opportunities to bring specialized tools into your workflows and deliver smarter, more dynamic solutions.

Exercise - Connect MCP tools to Azure AI Agents

Connect AI agents to tools using Model Context Protocol (MCP)

In this exercise, you'll create an agent that can connect to an MCP server and automatically discover callable functions.

You'll build a simple inventory assessment agent for a cosmetics retailer. Using the MCP server, the agent will be able to retrieve information about the inventory and make restock or clearance suggestions.

Module assessment

1. What role does the MCP server play in the MCP agent tool integration? **Hosts tool definitions and makes them available for discovery by the client.**
2. How does an MCP client retrieve available tools from the MCP server? **By calling `session.list_tools()` to get the current tool catalog..**
3. Why should MCP tools be wrapped in async functions on the client-side? **To enable asynchronous invocation so the agent can call tools without blocking.**

Summary

In this module, you learned how to **integrate external tools with Azure AI Foundry Agent Service using the Model Context Protocol (MCP)**.

By connecting your agent to an MCP server, you can **dynamically discover and register tools at runtime without hardcoding APIs or redeploying your agent. Using an MCP client, you generated function wrappers from discovered tools and connected them directly to your agent.** This integration allows your agent to adapt to evolving toolsets, and create more flexible AI solutions that can grow alongside your applications.

Tip: To learn more about MCP, see the [Model Context Protocol User Guide](#) and [AI Agents MCP Integration](#). To learn more about using Azure AI Foundry Agent Service with MCP, visit [Connect to Model Context Protocol servers](#).

Develop an AI agent with Microsoft Agent Framework

This module provides engineers with the skills to begin **building Azure AI Foundry Agent Service agents with Microsoft Agent Framework**.

Learning objectives

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

- Use **Microsoft Agent Framework** to connect to an Azure AI Foundry project.
- Create Azure AI Foundry Agent Service agents using the **Microsoft Agent Framework SDK**.
- Integrate plugin functions with your AI agent.

Introduction

AI agents are transforming how applications interact with users and automate tasks. Unlike traditional programs, AI agents use generative AI to interpret data, make decisions, and complete tasks with minimal human intervention. These agents use large language models to streamline complex workflows, making them ideal for automating business processes.

Developers can build AI agents using different tools, including the **Microsoft Agent Framework**. This open-source SDK simplifies the integration of AI models into applications. The Microsoft Agent Framework **supports different types of agents from multiple providers**, including **Azure AI Foundry, Azure OpenAI, OpenAI, Microsoft Copilot Studio, and Anthropic agents**. This module focuses on Azure AI Foundry Agents, which provide enterprise-grade capabilities using the Azure AI Foundry Agent Service.

Azure AI Foundry Agent Service is a fully managed service that enables developers to securely build, deploy, and scale high-quality extensible AI agents. Using the Foundry Agent Service, developers don't need to manage the underlying compute or storage resources. The Microsoft Agent Framework enables developers to quickly build agents on the Foundry Agent Service, supporting natural language processing and providing access to built-in tools in just a few lines of code.

While Foundry Agent Service provides a powerful foundation for building AI agents, the **Microsoft Agent Framework offers more flexibility and scalability**. If your solution requires multiple types of agents, using the Microsoft Agent Framework ensures consistency across your implementation. Finally, if you're planning to develop multi-agent solutions, the framework's workflow orchestration features allow you to coordinate collaborative agents efficiently—a topic covered in more detail in a later module.

Suppose you need to develop an AI agent that automatically formats and emails expense reports for employees. Your AI agent can extract data from submitted expense reports, format them correctly, and send them to the appropriate recipients when you use the Microsoft Agent Framework. The tools and functions feature allows your AI agent to interact with APIs, retrieve necessary data, and complete tasks.

In this module, you learn about the core features of the **Microsoft Agent Framework SDK**. You also learn how to create your own AI agents and extend their capabilities with tool functions.

After completing this module, you're now able to:

- Use the Microsoft Agent Framework to connect to an Azure AI Foundry project.
- Create Azure AI Foundry agents using the Microsoft Agent Framework.
- Integrate tool functions with your AI agent.

Understand Microsoft Agent Framework AI agents

An **AI agent** is a program that uses generative AI to interpret data, make decisions, and perform tasks on behalf of users or other applications. AI agents rely on large language models to perform their tasks. Unlike conventional programs, AI agents can function autonomously, handling complex workflows and automating processes without requiring continuous human oversight.

AI Agents can be developed using many different tools and platforms, including the **Microsoft Agent Framework**. The Microsoft Agent Framework is an open-source SDK that enables developers to easily integrate the latest AI models into their applications. **This framework provides a comprehensive foundation for creating functional agents that can use natural language processing to complete tasks and collaborate with other agents.**

Microsoft Agent Framework core components

The Microsoft Agent Framework offers different components that can be used individually or combined.

- **Chat clients** - provide abstractions for connecting to AI services from different providers under a common interface. Supported providers include Azure OpenAI, OpenAI, Anthropic, and more through the `BaseChatClient` abstraction.
- **Function tools** - containers for custom functions that extend agent capabilities. Agents can automatically invoke functions to integrate with external APIs and services.
- **Built-in tools** - prebuilt capabilities including **Code Interpreter** for Python execution, **File Search** for document analysis, and **Web Search** for internet access.
- **Conversation management** - structured message system with roles (USER, ASSISTANT, SYSTEM, TOOL) and `AgentThread` for persistent conversation context across interactions.
- **Workflow orchestration** - supports sequential workflows, concurrent execution, group chat, and handoff patterns for complex multi-agent collaboration.

The Microsoft Agent Framework helps streamline the creation of agents and allows multiple agents to work together in conversations while including human input. The framework supports different types of agents from multiple providers, including Azure AI Foundry, Azure OpenAI, OpenAI, Microsoft Copilot Studio, and Anthropic agents.

What is an Azure AI Foundry Agent?

Azure AI Foundry Agents provide enterprise-level capabilities using the Azure AI Foundry Agent Service. These agents offer advanced features for complex enterprise scenarios. Key benefits include:

- **Enterprise-level capabilities** – Built for Azure environments with advanced AI features including **code interpreter, function tools integration, and Model Context Protocol (MCP)** support.
- **Automatic tool invocation** – Agents can automatically call and execute tools, integrating seamlessly with Azure AI Search, Azure Functions, and other Azure services.
- **Thread and conversation management** – Provides built-in mechanisms for managing persistent conversation states across sessions, ensuring smooth multi-agent interactions.
- **Secure enterprise integration** – Enables secure and compliant AI agent development with **Azure CLI authentication, RBAC, and customizable storage** options.

When you use Azure AI Foundry Agents, you get the full power of enterprise Azure capabilities combined with the features of the Microsoft Agent Framework. These features can help you create robust AI-driven workflows that can scale efficiently across business applications.

Agent framework core concepts

- **BaseAgent** - the foundation for all agents with consistent methods, providing a unified interface across all agent types.
- **Agent threads** - manage persistent conversation context and store conversation history across sessions using the `AgentThread` class.
- **Chat messages** - organized structure for agent communication using role-based messaging (**USER, ASSISTANT, SYSTEM, TOOL**) that enables smooth communication and integration.
- **Workflow orchestration** - supports sequential workflows, running multiple agents in parallel, group conversations between agents, and transferring control between specialized agents.
- **Multi-modal support** - allows agents to work with text, images, and structured outputs, including vision capabilities and type-safe response generation.
- **Function tools** - let you add custom capabilities to agents by including custom functions with automatic schema generation from Python functions.
- **Authentication methods** - supports multiple authentication methods including **Azure CLI credentials, API keys, MSAL for Microsoft business authentication, and role-based access control**.

This framework supports autonomous, multi-agent AI behaviors while maintaining a flexible architecture that lets you mix and match agents, tools, and workflows as needed. The design lets you switch between OpenAI, Azure OpenAI, Anthropic, and other providers without changing your code, making it easy to build AI systems—from simple chatbots to complex business solutions.

Create an Azure AI agent with Microsoft Agent Framework

Azure AI Foundry Agent is a specialized agent within the Microsoft Agent Framework, designed to provide enterprise-level conversational capabilities with seamless tool integration. It automatically handles tool calling, so you don't need to manually parse and invoke functions. The agent also securely manages conversation history using threads, which reduces the work of maintaining state. The **Azure AI Foundry Agent supports many built-in tools**, including **code interpreter, file search, and web search**. It also provides integration capabilities for Azure AI Search, Azure Functions, and other Azure services.

Creating an AzureAI Agent

An Azure AI Foundry Agent includes all the core capabilities you typically need for enterprise AI applications, like function execution, planning, and memory access. This agent acts as a self-contained runtime with enterprise-level features.

To use an Azure AI Foundry Agent:

1. Create an Azure AI Foundry project.
2. Add the project connection string to your Microsoft Agent Framework application code.
3. Set up authentication credentials.
4. Create a `ChatAgent` with an `AzureAI-AgentClient`.
5. Define tools and instructions for your agent.

Here's the code that shows how to create an Azure AI Foundry Agent:

```
from agent_framework import AgentThread, ChatAgent
from agent_framework.azure import AzureAI-AgentClient
from azure.identity.aio import AzureCliCredential

def get_weather(
    location: Annotated[str, Field(description="The location to get the weather for.")],
) -> str:
    """Get the weather for a given location."""
    return f"The weather in {location} is sunny with a high of 25°C."

# Create a ChatAgent with Azure AI client {#create-a-chatagent-with-azure-ai-client }
async with (
    AzureCliCredential() as credential,
    ChatAgent(
        chat_client=AzureAI-AgentClient(async_credential=credential),
        instructions="You are a helpful weather agent.",
        tools=get_weather,
    ) as agent,
):
    # Agent is now ready to use
```

Once your agent is created, you can **create a thread to interact with your agent** and get responses to your questions. For example:

```
# Create the agent thread for ongoing conversation {#create-the-agent-thread-for-ongoing-conversation}
thread = agent.get_new_thread()

# Ask questions and get responses {#ask-questions-and-get-responses }
first_query = "What's the weather like in Seattle?"
print(f"User: {first_query}")
first_result = await agent.run(first_query, thread=thread)
print(f"Agent: {first_result.text}")
```

Azure AI Foundry Agent key components

The Microsoft Agent Framework Azure AI Foundry Agent uses the following components to work:

- **AzureAIAgentClient** - **manages the connection to your Azure AI Foundry project.** This client lets you access the services and models associated with your project and provides enterprise-level authentication and security features.
- **ChatAgent** - the **main agent** class that combines the client, instructions, and tools to create a working AI agent that can handle conversations and complete tasks.
- **AgentThread** - automatically keeps track of **conversation history between agents and users**, and manages the conversation state. You can create new threads or reuse existing ones to maintain context across interactions.
- **Tools integration** - support for custom functions that extend agent capabilities. Functions are automatically registered and can be called by agents to connect with external APIs and services.
- **Authentication credentials** - supports **Azure CLI credentials, service principal authentication, and other Azure identity options** for secure access to Azure AI services.
- **Thread management** - provides flexible options for thread creation, including automatic thread creation for simple scenarios and explicit thread management for ongoing conversations.

These components work together to let you create enterprise-level agents with instructions to define their purpose and get responses from AI models while maintaining security, scalability, and conversation context for business applications.

Add tools to Azure AI agent

In the Microsoft Agent Framework, tools allow your AI agent to use existing APIs and services to perform tasks it couldn't do on its own. Tools work through function calling, allowing AI to automatically request and use specific functions. The framework routes the request to the appropriate function in

your codebase and returns the results back to the large language model (LLM) so it can generate a final response.

To enable automatic function calling, tools need to provide details that describe how they work. The function's input, output, and purpose should be described in a way that the AI can understand, otherwise, the AI can't call the function correctly.

How to use tools with Azure AI Foundry Agent

The Microsoft Agent Framework supports both **custom function tools** and **built-in tools** that are ready to use out of the box.

Built-in tools

Azure AI Foundry Agents come with several built-in tools that you can use immediately:

- **Code Interpreter** - executes Python code for calculations, data analysis, and more
- **File Search** - searches through and analyzes documents
- **Web Search** - retrieves information from the internet

These tools are automatically available and don't require any extra setup.

Custom function tools

When creating custom tools for your Azure AI Foundry Agent, you need to understand several key concepts:

1. Function definition and annotations

Create your tool by defining a regular Python function with proper type annotations. Use Annotated and Field from Pydantic to provide detailed descriptions that help the AI understand the function's purpose and how to use its parameters. The more descriptive your annotations, the better the AI can understand when and how to call your function.

2. Adding tools to your agent

Pass your custom functions to the `ChatAgent` during creation using the `tools` parameter. You can add a single function or a list of multiple functions. The framework automatically registers these functions and makes them available for the AI to call.

3. Tool invocation through conversation

Once your tools are registered with the agent, you don't need to manually invoke them. Instead, ask the agent questions or give it tasks that would naturally require your tool's functionality. The AI automatically determines when to call your tools based on the conversation context and the tool descriptions you provided.

4. Multiple tools and orchestration

You can **add multiple tools to a single agent**, and the AI automatically chooses which tool to use based on the user's request. The framework handles the orchestration, calling the appropriate functions and combining their results to provide a comprehensive response.

Best practices for tool development

- **Clear descriptions:** Write clear, detailed descriptions for your functions and parameters to help the AI understand their purpose
- **Type annotations:** Use proper Python type hints to specify expected input and output types
- **Error handling:** Implement appropriate error handling in your tool functions to gracefully handle unexpected inputs
- **Return meaningful data:** Ensure your functions return data that the AI can effectively use in its responses
- **Keep functions focused:** Design each tool to handle a specific task rather than trying to do too many things in one function

By following these concepts, you can extend your Azure AI Foundry Agent with both built-in and custom tools, allowing it to interact with APIs and perform advanced tasks. This approach makes your AI more powerful and capable of handling real-world applications efficiently.

Exercise - Develop an Azure AI agent with the Microsoft Agent Framework SDK

Now you're ready to build an agent with the Microsoft Agent Framework. In this exercise, you use the Microsoft Agent Framework SDK to create an AI agent that creates an expense claim email.

Develop an Azure AI chat agent with the Microsoft Agent Framework SDK

In this exercise, you'll use Azure AI Agent Service and Microsoft Agent Framework to create an AI agent that processes expense claims.

Knowledge check

1. What are the key steps to create an AzureAIAgent? Create an `AzureAIAgentClient`, define a `ChatAgent` with instructions and tools, and create an `AgentThread` for conversations.
2. Which component in the Agent Framework manages conversation state and stores messages?
`AgentThread`
3. Which step is necessary to enable an AzureAIAgent to use a plugin? Create Python functions with proper type annotations and descriptions, then pass them to the `ChatAgent`'s `tools` parameter.

Summary

In this module, you learned how the Microsoft Agent Framework enables developers to build AI agents. You learned about the components and core concepts of the **Microsoft Agent Framework**. You also learned how to create custom tools to extend your agent's capabilities. By applying these concepts and skills, you can use the Microsoft Agent Framework to create dynamic, adaptable AI solutions that enhance user interactions and automate complex tasks.

Orchestrate a multi-agent solution using the Microsoft Agent Framework

Learn how to use the **Microsoft Agent Framework SDK** to develop your own AI agents that can collaborate for a multi-agent solution.

Learning objectives

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

- Build AI agents using the Microsoft Agent Framework SDK
- Understand **how and when to use different orchestration patterns**
- Develop multi-agent solutions

Introduction

AI agents offer a powerful combination of technologies, able to complete tasks with the use of generative AI. However, in some situations, the task required might be larger than is realistic for a single agent. For those scenarios, consider a **multi-agent solution**. A multi-agent solution allows agents to collaborate within the same conversation.

Imagine you're trying to address common DevOps challenges such as monitoring application performance, identifying issues, and deploying fixes. A multi-agent system could consist of four specialized agents working collaboratively:

- The **Monitoring Agent** continuously ingests logs and metrics, detects anomalies using natural language processing (NLP), and triggers alerts when issues arise.
- The **Root Cause Analysis Agent** then correlates these anomalies with recent system changes, using machine learning models or predefined rules to pinpoint the root cause of the problem.
- Once the root cause is identified, the **Automated Deployment Agent** takes over to implement fixes or roll back problematic changes by interacting with CI/CD pipelines and executing deployment scripts.
- Finally, the **Reporting Agent** generates detailed reports summarizing the anomalies, root causes, and resolutions, and notifies stakeholders via email or other communication channels.

This modular, scalable, and intelligent multi-agent system streamlines the DevOps process. The agents collaborate to reduce manual intervention and improve efficiency while ensuring timely communication and resolution of issues.

In this module, you'll explore how to use the powerful capabilities of the Microsoft Agent Framework to design and orchestrate intelligent agents that work collaboratively to solve complex problems. You'll also learn about the **different types of orchestration patterns** available, and use the Microsoft Agent Framework to develop your own AI agents that can collaborate for a multi-agent solution.

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

- Build AI agents using the **Microsoft Agent Framework SDK**
- Use tools and plugins with your AI agents
- Understand different types of orchestration patterns
- Develop multi-agent solutions

Understand the Microsoft Agent Framework

The Microsoft Agent Framework is an open-source SDK that enables developers to integrate AI models into their applications. This framework provides comprehensive support for creating AI-powered agents that can work independently or collaborate with other agents to accomplish complex tasks.

What is the Microsoft Agent Framework?

The Microsoft Agent Framework is designed to help developers build AI-powered agents that can process user inputs, make decisions, and execute tasks autonomously by leveraging large language models and traditional programming logic. The framework provides structured components for defining AI-driven workflows, enabling agents to interact with users, APIs, and external services seamlessly.

Core concepts

The Microsoft Agent Framework provides a flexible architecture with the following key components:

- **Agents:** Agents are intelligent, AI-driven entities capable of reasoning and executing tasks. They use large language models, tools, and conversation history to make decisions dynamically and respond to user needs.

- **Agent orchestration:** Multiple agents can collaborate towards a common goal using different **orchestration patterns**. The Microsoft Agent Framework supports several orchestration patterns with a unified interface for construction and invocation, allowing you to easily switch between patterns without rewriting your agent logic.

The framework includes several core features that power agent functionality:

- **Chat clients:** Chat clients provide abstractions for connecting to AI services from different providers under a common interface. Supported providers include Azure OpenAI, OpenAI, Anthropic, and more through the `BaseChatClient` abstraction.
- **Tools and function integration:** Tools enable agents to extend their capabilities through custom functions and built-in services. Agents can automatically invoke tools to integrate with external APIs, execute code, search files, or access web information. The framework supports both **custom function tools and built-in tools** like *Code Interpreter, File Search, and Web Search*.
- **Conversation management:** Agents can maintain conversation history across multiple interactions using `AgentThread`, allowing them to track previous interactions and adapt responses accordingly. The structured message system uses **roles (USER, ASSISTANT, SYSTEM, TOOL)** for persistent conversation context.

Types of agents

The Microsoft Agent Framework supports several different types of agents from multiple providers:

- **Azure AI Foundry Agent** - a specialized agent within the Microsoft Agent Framework designed to provide enterprise-grade conversational capabilities with seamless tool integration. It automatically handles tool calling and securely **manages conversation history using threads**, reducing the overhead of maintaining state. Azure AI Foundry Agents support built-in tools and provide integration capabilities for Azure AI Search, Azure Functions, and other Azure services.
- **ChatAgent**: designed for general conversation and task completion interfaces. The `ChatAgent` type provides natural language processing, contextual understanding, and dialogue management with support for custom tools and instructions.
- **OpenAI Assistant Agent**: designed for advanced capabilities using **OpenAI's Assistant API**. This agent type supports goal-driven interactions with features like code interpretation and file search through the OpenAI platform.
- **Anthropic Agent**: provides access to **Anthropic's Claude models** with the framework's unified interface, supporting advanced reasoning and conversation capabilities.

Why you should use the Semantic Kernel Agent Framework

The Microsoft Agent Framework offers a robust platform for building intelligent, autonomous, and collaborative AI agents. **The framework can integrate agents from multiple sources, including Azure AI Foundry Agent Service, and supports both multi-agent collaboration and human-agent interaction.** Agents can work together to orchestrate sophisticated workflows, where each agent specializes in a specific task, such as data collection, analysis, or decision-making. The framework also facilitates human-in-the-loop processes, enabling agents to augment human decision-making by providing insights or automating repetitive tasks. The provider-agnostic design allows you to switch between different AI providers without changing your code, making it suitable for building adaptable AI systems from simple chatbots to complex enterprise solutions.

Understand agent orchestration

The **Microsoft Agent Framework SDK's agent orchestration framework** makes it possible to design, manage, and scale complex multi-agent workflows without having to manually handle the details of agent coordination. Instead of relying on a single agent to manage every aspect of a task, you can combine multiple specialized agents. Each agent with a unique role or area of expertise can collaborate to create systems that are more robust, adaptive, and capable of solving real-world problems collaboratively.

By orchestrating agents together, you can take on tasks that would be too complex for a single agent—from running parallel analyses, to building multi-stage processing pipelines, to managing dynamic, context-driven handoffs between experts.

Why multi-agent orchestration matters

Single-agent systems are often limited in scope, constrained by one set of instructions or a single model prompt. Multi-agent orchestration addresses this limitation by allowing you to:

- **Assign** distinct skills, responsibilities, or perspectives to each agent.
- **Combine outputs** from multiple agents to improve decision-making and accuracy.
- **Coordinate steps in a workflow** so each agent's work builds on the last.
- **Dynamically route control** between agents based on context or rules.

This approach opens the door to more flexible, efficient, and scalable solutions, especially for real-world applications that require collaboration, specialization, or redundancy.

Supported orchestration patterns

Microsoft Agent Framework provides several orchestration patterns directly in the SDK, each offering a different approach to coordinating agents. These patterns are designed to be technology-agnostic so you can adapt them to your own domain and integrate them into your existing systems.

- **Concurrent orchestration** - Broadcast the same task to multiple agents at once and collect their results independently. Useful for **parallel analysis**, independent subtasks, or ensemble decision making.
- **Sequential orchestration** - Pass the output from one agent to the next in a fixed order. Ideal for **step-by-step** workflows, pipelines, and progressive refinement.
- **Handoff orchestration** - **Dynamically transfer control** between agents based on context or rules. Great for escalation, fallback, and expert routing where one agent works at a time.
- **Group chat orchestration** - Coordinate a shared conversation among multiple agents (and optionally a human), managed by a chat manager that chooses who speaks next. Best for **brainstorming**, collaborative problem solving, and building consensus.
- **Magnetic orchestration** - A manager-driven approach that plans, delegates, and adapts across specialized agents. Suited to **complex, open-ended problems** where the solution path evolves.

A unified orchestration workflow

Regardless of which orchestration pattern you choose, the Microsoft Agent Framework SDK provides a consistent, developer-friendly interface for building and running them. The typical flow looks like this:

1. **Define your agents** and describe their capabilities.
2. **Select and create an orchestration pattern**, optionally adding a manager agent if needed.
3. **Optionally configure callbacks or transforms** for custom input and output handling.
4. **Start a runtime** to manage execution.
5. **Invoke the orchestration** with your task.
6. **Retrieve results** in an **asynchronous**, nonblocking way.

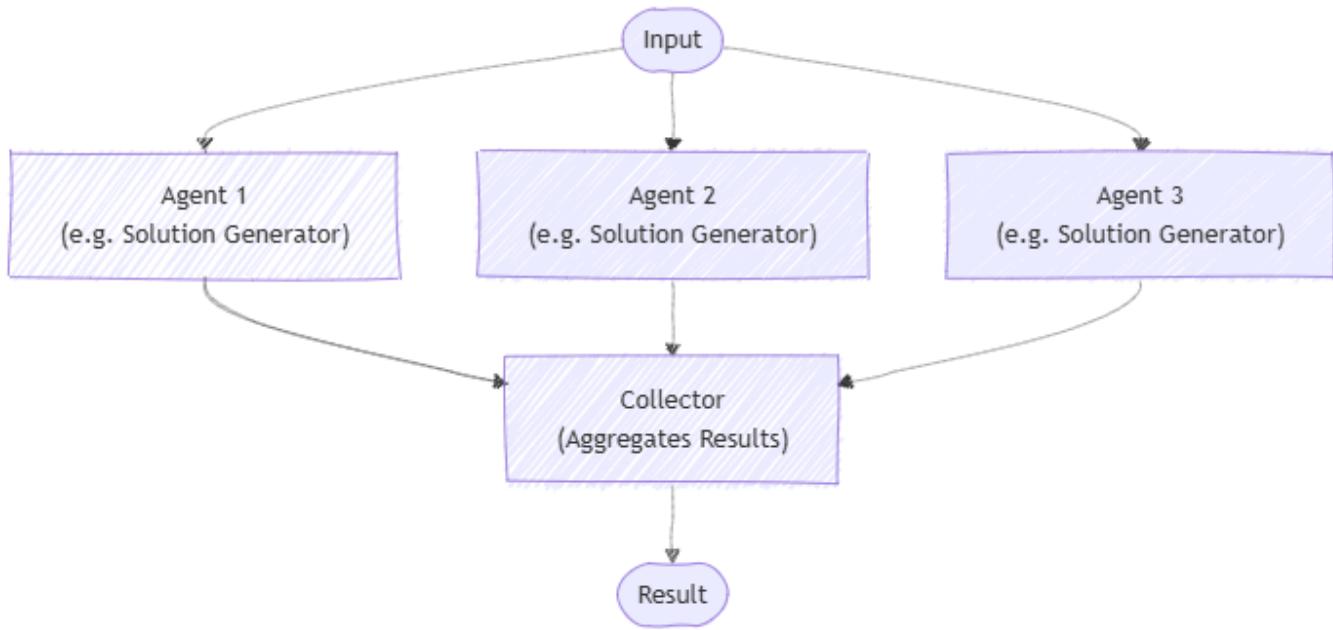
Because all patterns share the same core interface, you can easily experiment with different orchestration strategies without rewriting agent logic or learning new APIs. The SDK abstracts the complexity of agent communication, coordination, and result aggregation so you can focus on designing workflows that deliver results.

Multi-agent orchestration in the Microsoft Agent Framework SDK provides a flexible, scalable way to build intelligent systems that combine the strengths of multiple specialized agents. With built-in orchestration patterns, a unified development model, and runtime features for managing execution, you can quickly prototype, refine, and deploy collaborative AI workflows. The framework provides the

tools to turn multiple agents into a cohesive problem-solving team, whether you're running parallel processes, sequential workflows, or dynamic conversations.

Use concurrent orchestration

Concurrent orchestration lets multiple agents work on the same task at the same time. **Each agent handles the task independently, and then their outputs are gathered and combined.** This method works especially well when you want diverse approaches or solutions, like during **brainstorming, group decision-making, or voting.**



This pattern is useful when you need different approaches or ideas to solve the same problem. Instead of having agents work one after another, they **all work at the same time**. This speeds up the process and covers the problem from many angles.

Usually, the results from each agent are combined to create a final answer, but this isn't always necessary. Each agent can also produce its own separate result, like calling tools to complete tasks or updating different data stores independently.

Agents work on their own and don't share results with each other. However, an agent can call other AI agents by running its own orchestration as part of its process. Agents need to know which other agents are available to work on tasks. This pattern allows you to either call all registered agents every time or choose which agents to run based on the specific task.

When to use concurrent orchestration

You may want to consider using the concurrent orchestration pattern in these situations:

- When tasks can run at the same time, either by using a fixed group of agents or by selecting AI agents dynamically based on what the task needs.
- When the task benefits from different specialized skills or approaches (for example, technical, business, or creative) that all work independently but contribute to solving the same problem.

This kind of teamwork is common in multi-agent decision-making methods such as:

- Brainstorming ideas
- Combining different reasoning methods (ensemble reasoning)
- Making decisions based on voting or consensus (quorum)
- Handling tasks where speed matters and running agents in parallel cuts down wait time

When to avoid concurrent orchestration

You may want to avoid using the concurrent orchestration pattern in the following scenarios:

- Agents need to build on each other's work or **depend on shared context** in a specific order.
- The task requires a **strict sequence** of steps or predictable, repeatable results.
- Resource limits, like model usage quotas, make running agents in parallel inefficient or impossible.
- Agents can't reliably coordinate changes to shared data or external systems while running at the same time.
- There's **no clear way to resolve conflicts or contradictions** between results from different agents.
- Combining results is too complicated or ends up lowering the overall quality.

Implement concurrent orchestration

Implement the concurrent orchestration pattern with the Microsoft Agent Framework:

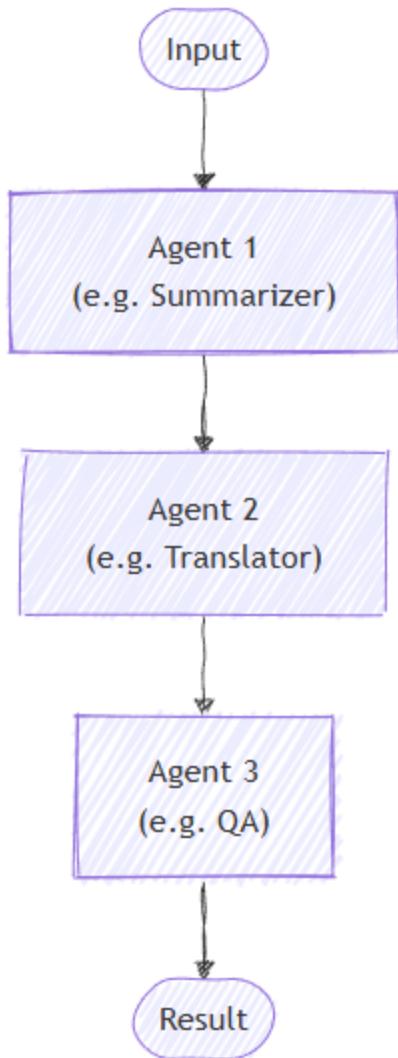
1. **Create your chat client:** Set up a chat client (for example, `AzureOpenAIChatClient`) with appropriate credentials to connect to your AI service provider.
2. **Define your agents:** Create agent instances using the chat client's `create_agent` method. Each agent should have specific instructions and a name that defines its role and expertise area.
3. **Build the concurrent workflow:** Use the `ConcurrentBuilder` class to create a workflow that can run multiple agents in parallel. Add your agent instances as participants using the `participants()` method, then call `build()` to create the workflow.

4. **Run the workflow:** Call the workflow's `run` method with the task or input you want the agents to work on. The workflow runs all agents concurrently and returns events containing the results.
5. **Process the results:** Extract the outputs from the workflow events using `get_outputs()`. The results contain the combined conversations from all agents, with each agent's response included in the final output.
6. **Handle the aggregated responses:** Process the aggregated messages from all agents. **Each message includes the author name and content**, allowing you to identify which agent provided each response.

Concurrent orchestration is a powerful pattern for using multiple AI agents **simultaneously**, enabling faster and more diverse problem-solving. By running agents in parallel, you can explore different approaches at once, improve efficiency, and gain richer insights. However, it's important to choose this pattern when tasks can truly run independently and to be mindful of resource constraints and coordination challenges. When implemented thoughtfully with the Microsoft Agent Framework SDK, concurrent orchestration can greatly enhance your AI workflows and decision-making processes.

Use sequential orchestration

In sequential orchestration, agents are arranged in a **pipeline** where each agent processes the task one after another. **The output from one agent becomes the input for the next**. This pattern is ideal for workflows where each step depends on the previous one, such as document review, data transformation pipelines, or multi-stage reasoning.



Sequential orchestration works best for tasks that need to be done **step-by-step**, with each step improving on the last. The order in which agents run is fixed and decided beforehand, and agents don't decide what happens next.

When to use sequential orchestration

Consider using the sequential orchestration pattern when your workflow has:

- Processes made up of **multiple steps** that must happen in a **specific order**, where each step relies on the one before it.
- Data workflows where each stage adds something important that the next stage needs to work properly.
- Tasks where stages can't be done at the same time and must run one after another.
- Situations that require **gradual improvements**, like drafting, reviewing, and polishing content.

- Systems where you know how each agent performs and can handle delays or failures in any step without stopping the whole process.

When to avoid sequential orchestration

Avoid this pattern when:

- **Stages can be run independently and in parallel** without affecting quality.
- A single agent can perform the entire task effectively.
- Early stages may fail or produce poor output, and there's no way to stop or correct downstream processing based on errors.
- Agents need to collaborate dynamically rather than hand off work sequentially.
- The workflow requires iteration, backtracking, or dynamic routing based on intermediate results.

Implement sequential orchestration

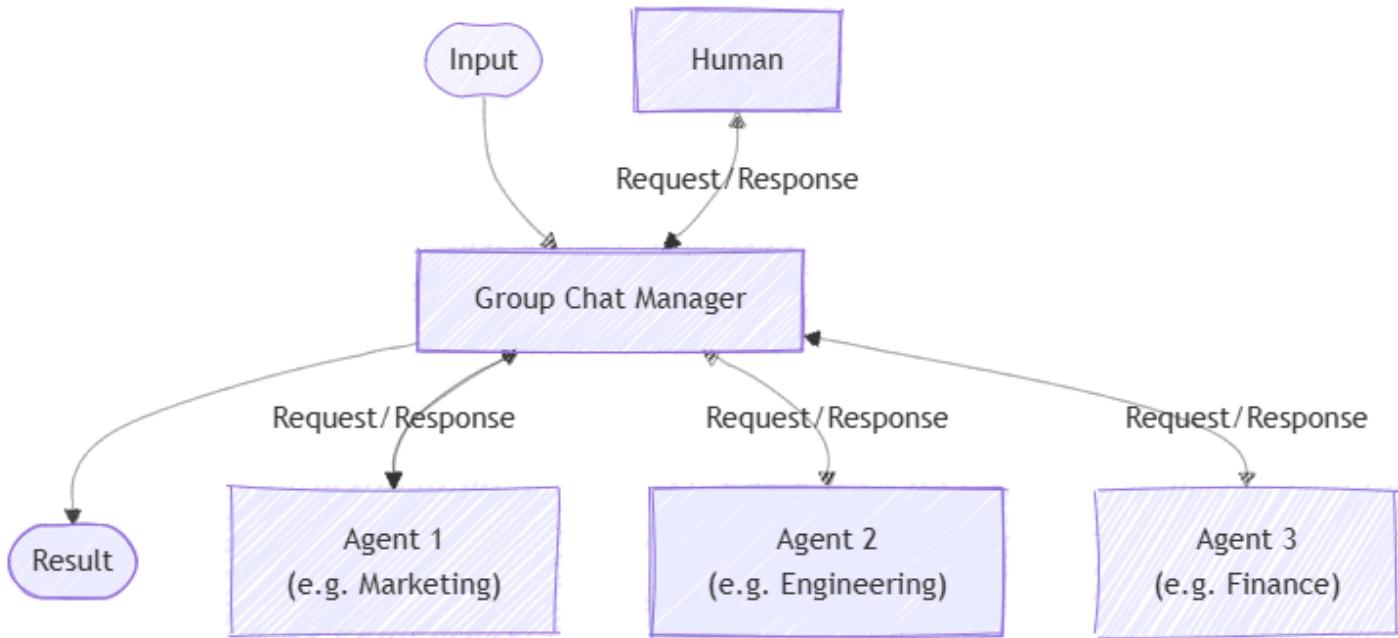
Implement the sequential orchestration pattern with the Microsoft Agent Framework SDK:

1. **Create your chat client:** Set up a chat client (for example, `AzureOpenAIChatClient`) with appropriate credentials to connect to your AI service provider.
2. **Define your agents:** Create agent instances using the chat client's `create_agent` method. Each agent should have specific instructions and a name that defines its role and expertise area in the pipeline.
3. **Build the sequential workflow:** Use the `SequentialBuilder` class to create a workflow that executes agents one after another. Add your agent instances as participants using the `participants()` method, then call `build()` to create the workflow.
4. **Run the workflow:** Call the workflow's `run_stream` method with the task or input you want the agents to work on. The workflow processes the task through all agents sequentially, with each agent's output becoming input for the next.
5. **Process the workflow events:** Iterate through the workflow events using an async loop. Look for `WorkflowOutputEvent` instances, which contain the results from the sequential processing.
6. **Extract the final conversation:** Collect the final conversation from the workflow outputs. The result contains the complete conversation history showing how each agent in the sequence contributed to the final outcome.

Sequential orchestration is ideal when your task requires clear, ordered steps where each agent builds on the previous one's output. This pattern helps improve output quality through stepwise refinement and ensures predictable workflows. When applied thoughtfully with the Microsoft Agent Framework SDK, it enables powerful multi-agent pipelines for complex tasks like content creation, data processing, and more.

Use group chat orchestration

Group chat orchestration models a collaborative conversation among multiple AI agents, and optionally a human participant. A central chat manager controls the flow, deciding which agent responds next and when to request human input. This pattern is useful for simulating meetings, debates, or collaborative problem-solving.



The group chat pattern works well for scenarios where **group discussion** or **iterative collaboration** is key to reaching decisions. It supports different interaction styles, from free-flowing ideation to formal workflows with defined roles and approval steps. Group chat orchestration is also great for **human-in-the-loop** setups where a human may guide or intervene in the conversation. Typically, agents in this pattern don't directly change running systems—they mainly contribute to the conversation.

When to use group chat orchestration

Consider using group chat orchestration when your scenario involves:

- Spontaneous or guided collaboration among agents (and possibly humans)
- Iterative maker-checker loops where agents take turns creating and reviewing
- **Real-time human oversight or participation**
- Transparent and auditable conversations since **all output is collected in a single thread**

Common scenarios include:

- Creative brainstorming where agents build on each other's ideas
- Decision-making that benefits from debate and consensus
- Complex problems requiring cross-disciplinary dialogue
- Quality control and validation requiring multiple expert perspectives
- Content workflows with clear separation between creation and review

When to avoid group chat orchestration

Avoid this pattern when:

- Simple task delegation or straightforward linear pipelines suffice
- Real-time speed requirements make discussion overhead impractical
- Hierarchical or deterministic workflows are needed without discussion
- The chat manager can't clearly determine when the task is complete
- Managing conversation flow becomes too complex, especially with many agents (limit to three or fewer for easier control)

Maker-checker loops

A common special case is the maker-checker loop. Here, **one agent (the maker) proposes content or solutions, and another agent (the checker) reviews and critiques them**. The checker can send feedback back to the maker, and this cycle repeats until the result is satisfactory. **This process requires a turn-based sequence managed by the chat manager.**

Implement group chat orchestration

Implement the group chat orchestration pattern with the Microsoft Agent Framework SDK:

- 1. Create your chat client:** Set up a chat client (for example, `AzureOpenAIChatClient`) with appropriate credentials to connect to your AI service provider.
- 2. Define your agents:** Create agent instances using the chat client's `create_agent` method. Each agent should have **specific instructions and a name** that defines its role and expertise area.
- 3. Build the group chat workflow:** Use the `GroupChatBuilder` class to create a workflow that can run multiple agents in parallel. Add your agent instances as participants using the `participants()` method, then call `build()` to create the workflow.
- 4. Run the workflow:** Call the workflow's `run` method with the task or input you want the agents to work on. The workflow runs all agents concurrently and returns events containing the results.
- 5. Process the results:** Extract the outputs from the workflow events using `get_outputs()`. The results contain the combined conversations from all agents, with each agent's response included in the final output.

6. Handle the aggregated responses: Process the aggregated messages from all agents. Each message includes the author name and content, allowing you to identify which agent provided each response.

Customizing the group chat manager

You can create a custom group chat manager by extending the base `GroupChatManager` class. This approach lets you control:

- How conversation results are filtered or summarized
- How the next agent is selected
- When to request user input
- When to terminate the conversation

Custom managers let you implement specialized logic tailored to your use case.

Group chat manager call order

During each round of the conversation, the chat manager calls methods in this order:

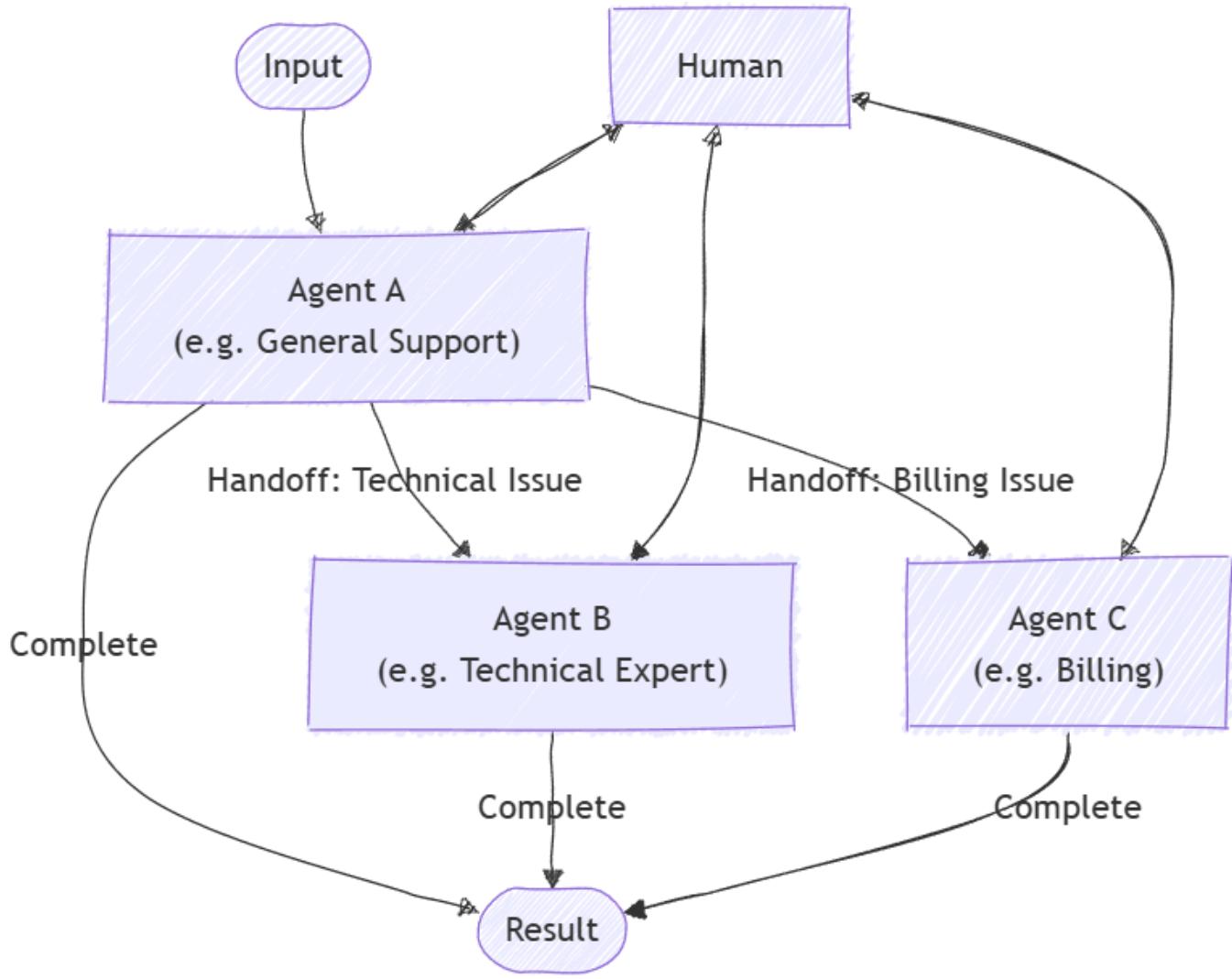
1. `should_request_user_input` - Checks if human input is needed before the next agent responds.
2. `should_terminate` - Determines if the conversation should end (for example, max rounds reached).
3. `filter_results` - If ending, summarizes or processes the final conversation.
4. `select_next_agent` - If continuing, chooses the next agent to speak.

This ensures user input and termination conditions are handled before moving the conversation forward. Override these methods in your custom manager to change behavior.

Group chat orchestration enables multiple AI agents—and optionally humans—to collaborate through guided conversation and iterative feedback. It's ideal for complex tasks that benefit from diverse expertise and dynamic interaction. While it requires careful management, this pattern offers transparency and flexibility in decision-making and creative workflows. The Microsoft Agent Framework SDK makes it easy to implement and customize group chat orchestration for your needs.

Use handoff orchestration

Handoff orchestration lets AI agents transfer control to one another based on the task context or user requests. Each agent can "handoff" the conversation to another agent with the right expertise, making sure the best-suited agent handles each part of the task. This pattern is ideal for customer support, expert systems, or any situation where dynamic delegation is needed.



This pattern fits scenarios where the best agent isn't known upfront or where the task requirements become clearer during processing. Unlike parallel patterns, agents work one at a time, fully handing off control from one to the next.

When to use handoff orchestration

You may want to consider using the handoff orchestration pattern in these scenarios:

- Tasks need specialized knowledge or tools, but the number or order of agents can't be determined in advance.
- Expertise requirements emerge dynamically during processing, triggering task routing based on content analysis.
- Multiple-domain problems require different specialists working sequentially.
- You can define clear signals or rules indicating when an agent should transfer control and to whom.

When to avoid handoff orchestration

You may want to avoid using the handoff orchestration pattern in these scenarios:

- The involved agents and their order are known upfront and fixed.
- Task routing is simple and rule-based, not needing dynamic interpretation.
- Poor routing decisions might frustrate users.
- Multiple operations must run at the same time.
- Avoiding infinite handoff loops or excessive bouncing between agents is difficult.

Implementing handoff orchestration

The handoff orchestration pattern can be implemented in the Microsoft Agent Framework SDK using control workflows. In a control workflow, each agent processes the task in sequence, and based on its output, the **workflow decides which agent to call next**. This routing is done using a **switch-case structure** that routes the task to different agents based on classification results.

1. Set up data models and chat client:

- Create your chat client for connecting to AI services
- Define Pydantic models for AI agents' structured JSON responses
- Create simple data classes for passing information between workflow steps
- Configure agents with specific instructions and `response_format` parameter for structured JSON output

2. Create specialized executor functions:

- **Input storage executor** - saves incoming data to shared state and forwards to classification agent
- **Transformation executor** - converts agent's JSON response into typed routing object
- **Handler executors** - separate executors for each classification outcome with guard conditions to verify correct message processing

3. Build routing logic:

- Create factory functions that generate condition checkers for each classification value

- Design conditions to examine incoming messages and return true for specific classification results
- Use conditions with Case objects in **switch-case edge groups**
- Always include **a Default case as fallback** for unexpected scenarios

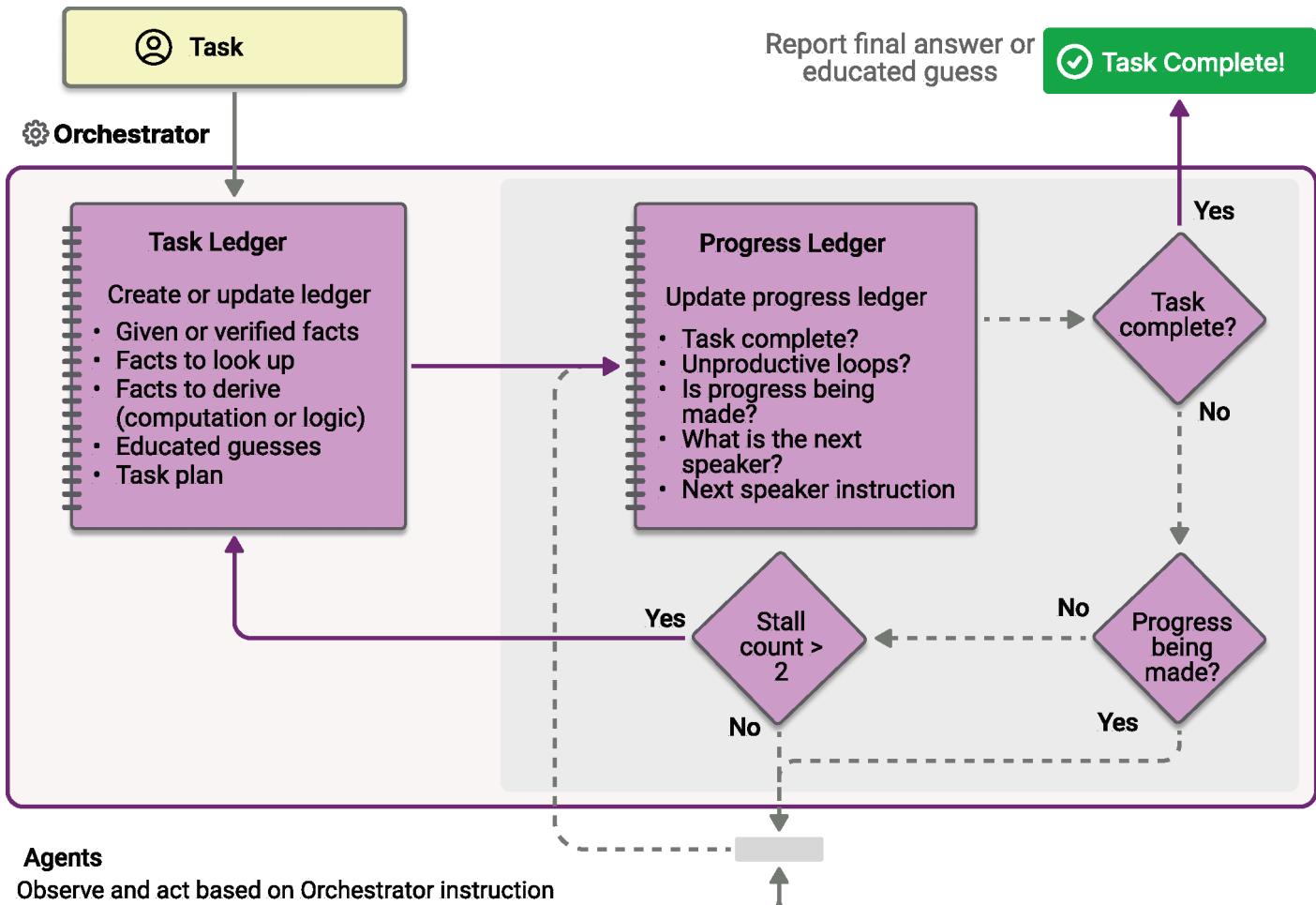
4. Assemble the workflow:

- Use WorkflowBuilder to connect executors with regular edges
- Add switch-case edge group for routing based on classification results
- Configure workflow to follow first matching case or fall back to default
- Set up **terminal executor** to yield final output

Handoff orchestration provides a flexible way to route tasks dynamically among specialized AI agents, ensuring that each part of a workflow is handled by the **best-suited expert**. It works well for complex, evolving tasks like **customer support or multi-domain problem solving** where expertise needs change during the conversation. When you use the Microsoft Agent Framework SDK, you can build adaptable systems that seamlessly transfer control between agents—and include human input when needed—for smooth and efficient task completion.

Use Magentic orchestration

Magnetic orchestration is a flexible, **general-purpose multi-agent pattern designed for complex, open-ended tasks that require dynamic collaboration**. This pattern uses a dedicated Magnetic manager to coordinate a team of specialized agents. The manager decides which agent should act next based on the evolving context, task progress, and agent capabilities.



The Magentic manager maintains a shared context, tracks progress, and adapts the workflow in real time. This approach allows the system to break down complex problems, assign subtasks, and iteratively refine solutions. The process focuses as much on building and documenting the approach as it does on delivering the final solution. A dynamic task ledger is built and refined as the workflow progresses, recording goals, subgoals, and execution plans.

When to use Magentic orchestration

Consider using the Magentic orchestration pattern in these scenarios:

- **The problem is complex or open-ended with no predetermined solution path.**
- Input and feedback from multiple specialized agents are needed to shape a valid solution.
- **The system must generate a documented plan of approach for human review.**
- Agents have tools that can directly interact with external systems and resources.
- A step-by-step, dynamically built execution plan adds value before running the tasks.

When to avoid Magentic orchestration

You may want to avoid this pattern when:

- The solution path is fixed or deterministic.
- There's no need to produce a ledger or plan of approach.
- The task is simple enough for a more lightweight orchestration pattern.
- Speed is the priority, as this method emphasizes planning over fast execution.
- You expect frequent stalls or loops without a clear resolution path.

Implementing Magentic orchestration

Implement the Magentic orchestration pattern with the Microsoft Agent Framework:

- 1. Define specialized agents:** Create agent instances (for example, `chatAgent`) with specific instructions and chat clients. Each agent should have a specialized role and capabilities suited for different aspects of the complex task.
- 2. Set up event handling callback:** Define an async callback function to handle different types of events during orchestration, including orchestrator messages, agent streaming updates, agent messages, and final results.
- 3. Build the Magentic workflow:** Use the `MagneticBuilder` class to create the orchestration. Add your agent instances as participants, configure the event callback with streaming mode, and set up the standard manager with appropriate parameters like `max round count` and `stall limits`.
- 4. Configure the standard manager:** The standard manager coordinates agent collaboration using a chat client for planning and progress tracking. Configure parameters like `maximum round count`, `stall count`, and `reset count` to control the orchestration behavior.
- 5. Run the workflow:** Call the workflow's `run_stream` method with your complex task. The workflow dynamically plans, delegates work to appropriate agents, and coordinates their collaboration to solve the problem.
- 6. Process workflow events:** Iterate through the workflow events using an async loop. Handle different event types including `WorkflowOutputEvent`, which contains the final results from the orchestration.
- 7. Extract the final result:** Collect the final output from the workflow events. The result contains the complete solution developed through the collaborative effort of all participating agents.

Magnetic orchestration excels at **solving complex, evolving problems that require real-time coordination between specialized agents**. It's ideal for **tasks where the plan can't be defined in advance and must adapt as new information emerges**. Using the Microsoft Agent Framework, you can build systems that dynamically design, refine, and execute solution paths through intelligent agent collaboration.

Exercise - Develop a multi-agent solution

Now it's your opportunity to **build a multi agent solution with the Semantic Kernel Agent**

Framework. In this exercise, you create an application that automatically triages and resolves issues presented in log files of a system. Using Azure AI Agents, you create an *incident manager agent* and a *devops agent* that collaborates to fix the issues.

Develop a multi-agent solution

In this exercise, you'll practice using the **sequential orchestration pattern** in the Microsoft Agent Framework SDK. You'll create a simple pipeline of three agents that work together to process customer feedback and suggest next steps. You'll create the following agents:

- The **Summarizer agent** will condense raw feedback into a short, neutral sentence.
- The **Classifier agent** will categorize the feedback as Positive, Negative, or a Feature request.
- Finally, the **Recommended Action agent** will recommend an appropriate follow-up step.

You'll learn how to use the Microsoft Agent Framework SDK to break down a problem, route it through the right agents, and produce actionable results.

Knowledge check

1. What's the first step in the Microsoft Agent Framework's unified orchestration workflow? Define your agents and describe their capabilities
2. For brainstorming and collaborative problem solving among multiple agents, which orchestration pattern is most suitable? Group Chat
3. Which pattern dynamically transfers control between agents based on context or rules? Handoff

Summary

In this module, you explored how to design and manage multi-agent orchestration workflows using the Microsoft Agent Framework SDK.

You learned how multi-agent systems provide advantages over single-agent approaches, including improved scalability, specialization, and collaborative problem solving. You also learned several

different orchestration patterns—**concurrent, sequential, handoff, group chat, and magentic**—and reviewed guidance on when and how to use each.

You also saw how the SDK provides a unified interface for defining agents, running orchestrations, handling structured data, and retrieving results asynchronously, enabling you to build flexible, reliable, and maintainable multi-agent workflows.

Discover Azure AI Agents with A2A

Learn how to implement the **A2A protocol** to enable **agent discovery, direct communication, and coordinated task execution** across remote agents.

Learning objectives

After completing this module, you're able to:

- Understand the A2A protocol and its role in multi-agent orchestration.
- Design **discoverable agents** for modular, collaborative problem-solving.
- Implement A2A strategies to discover and invoke remote agents.

Introduction

AI agents are powerful on their own, but many real-world tasks require **collaboration across multiple agents**. Coordinating these interactions manually can be complex, especially when agents are remote or distributed.

The **Agent-to-Agent (A2A) protocol** addresses this challenge by providing a standardized framework for agent discovery, communication, and coordinated task execution. By implementing A2A, you can easily manage connections to remote agents, delegate requests to the appropriate agent, and enable seamless communication between agents in a standardized, secure way.

For example, imagine a technical writer who wants to create compelling blog content. One agent generates compelling article headlines, while another creates detailed outlines. Using A2A, a routing agent coordinates the workflow: it sends the user's request to the title agent, passes the generated title to the outline agent, and returns the final outline to the user—all automatically.

In this module, you learn how to **implement the A2A protocol with Azure AI Agents**. You also practice configuring a routing agent, registering remote agents, and building a coordinated workflow that allows multiple agents to collaborate effectively.

Define an A2A agent

Before an A2A agent can participate in multi-agent workflows, it needs to explain what it can do. **Agent Skills and how other agents or clients can discover those capabilities are exposed through an Agent Card.**

Agent Skills

An Agent Skill describes a specific capability or function that the agent can perform. Think of it as a building block that communicates to clients or other agents what tasks the agent is designed to handle.

Key elements of an Agent Skill include:

- **ID:** A unique identifier for the skill.
- **Name:** A human-readable name describing the skill.
- **Description:** A detailed explanation of what the skill does.
- **Tags:** Keywords for categorization and easier discovery.
- **Examples:** Sample prompts or use cases to illustrate the skill in action.
- **Input/Output Modes:** Supported data formats or media types (for example, text, JSON).

When defining a skill for your agent, consider the tasks it should perform, how to describe them clearly, and how other agents or clients might use them. For example, a simple "Hello World" skill could return a basic greeting in text format, whereas a blog-writing skill might accept a topic and return a suggested title or outline.

Agent Card

The Agent Card is like a digital business card for your agent. It's a structured document that a routing agent or client can retrieve to discover your agent's capabilities and how to interact with it.

Key elements of an Agent Card include:

- **Identity Information:** Name, description, and version of the agent.
- **Endpoint URL:** Where the agent's A2A service can be accessed.
- **Capabilities:** Supported A2A features such as streaming or push notifications.
- **Default Input/Output Modes:** The primary media types the agent can handle.
- **Skills:** A list of the agent's skills that other agents can invoke.
- **Authentication Support:** Indicates if the agent requires credentials for access.

When creating an Agent Card, ensure it accurately represents your **agent's skills and endpoints**. This allows clients or routing agents to discover the agent, understand what it can do, and interact with it appropriately.

Putting it together

Once an agent defines its skills and publishes an Agent Card:

- Other agents or clients can discover the agent automatically.
- Requests can be routed to the agent's appropriate skill.
- Responses are returned in supported formats, enabling smooth collaboration across multiple agents.

For example, in a technical writer workflow, one agent could define skills for generating article titles, and another for creating outlines. The routing agent retrieves each agent's card to discover these capabilities and orchestrates a workflow where a title generated by one agent feeds into the outline agent, producing a cohesive final response.

Implement an agent executor

The Agent Executor is a core component of an A2A agent. **It defines how your agent processes incoming requests, generates responses, and communicates with clients or other agents**. Think of it as the bridge between the A2A protocol and your agent's specific business logic.

Understand the Agent Executor

The `AgentExecutor` interface **handles all incoming requests sent to your agent**. It receives information about the request, processes it according to the agent's capabilities, and sends responses or events back through a communication channel.

Key responsibilities:

- Execute tasks requested by users or other agents.
- Stream responses or send individual messages back to the client.
- Handle task cancellation if supported.

Implement the interface

An Agent Executor typically defines two primary operations:

Execute

- Processes incoming requests and generates responses.
- Accesses request details (for example, user input, task context).
- Sends results back via an event queue, which may include messages, task updates, or artifacts.

Cancel

- Handles requests to cancel an ongoing task.
- May not be supported for simple agents.

The executor uses the **RequestContext** to understand the incoming request and an **EventQueue** to communicate results or events back to the client.

Request handling flow

Consider a "Hello World" agent workflow:

1. The agent has a small helper class that implements its core logic (for example, returning a string).
2. The executor receives a request and calls the agent's logic.
3. The executor wraps the result as an event and places it on the event queue.
4. The routing mechanism sends the event back to the requester.

For cancellation, a basic agent might only indicate that cancellation isn't supported.

The Agent Executor is central to making your A2A agent functional. It defines how the agent executes tasks and communicates results, providing a standardized interface for clients and other agents. Properly implemented executors enable seamless integration and collaboration in multi-agent workflows.

Host an A2A server

Once your agent defines its skills and Agent Card, the next step is to host it on a server. **Hosting makes your agent accessible to clients and other agents over HTTP**, enabling real-time interactions and multi-agent workflows.

Hosting an agent allows it to:

- Expose its capabilities through its Agent Card, which clients and other agents can discover.
- Receive incoming A2A requests and forward them to your *Agent Executor* for processing.
- Manage task lifecycles, including streaming responses and stateful interactions.

Effectively, **the server acts as a bridge between your agent's logic and the external world**, ensuring it can participate in coordinated workflows.

Core components of the agent server

To host an agent, you need three essential components working together:

1. Agent Card

- Describes the agent's capabilities, skills, and input/output modes.
- Exposed at a standard endpoint (typically `./well-known/agent-card.json`) so clients and other agents can discover your agent.
- Can include multiple versions or an "extended" card for authenticated users.

2. Request Handler

- Routes incoming requests to the appropriate methods on your Agent Executor (for example, `execute` or `cancel`).
- Manages the task lifecycle using a Task Store, which tracks tasks, streaming data, and resubscriptions.
- Even simple agents require a task store to handle interactions reliably.

3. Server Application

- Built using a web framework (`Starlette` in Python) to handle HTTP requests.
- Combined with an ASGI server (like `Uvicorn`) to start listening on a network interface and port.
- Exposes the agent card and request handler endpoints, enabling clients to interact with your agent.

Set up the A2A agent server

1. **Define** your agent's skills and Agent Card.
2. **Initialize a request handler** that links your Agent Executor with a Task Store.
3. **Set up the server application**, providing the Agent Card and request handler.
4. **Start the server using an ASGI server (Uvicorn)** to make it accessible on the network.

- Once running, **the agent listens for incoming requests and responds** according to its defined skills.

A "Hello World" agent may expose a basic greeting skill. Once hosted, it can respond to any requests sent to its endpoint. A more complex agent can serve multiple skills or an extended Agent Card for authenticated users.

Hosting an A2A agent combines the **Agent Card**, **request handler**, and **agent executor** to make it available for client and agent interactions. This setup ensures tasks are managed correctly and responses are delivered reliably, enabling your agent to participate in multi-agent workflows.

Connect to your A2A agent

Once your A2A agent server is running, the next step is understanding how a client can interact with it. **A client acts as the bridge between your application and the agent server.**

The client responsibilities include:

- **Discovering the Agent Card**, which contains metadata about the agent and its endpoints.
- **Sending requests to the agent** for processing.
- **Receiving and interpreting the agent's responses**, which can be either direct messages or task-based results.

Connect to your agent server

- The client must know the base URL of the server.
- The client typically retrieves the Agent Card from a well-known endpoint on the server.
- Once the Agent Card is obtained, the client can be initialized with it, establishing a connection ready to send messages.

Send requests to the agent

There are two main types of requests a client can make:

- **Non-Streaming Requests**: The client sends a message and **waits for a complete response**. This type of request is suitable for simple interactions or when a single response is expected.
- **Streaming Requests**: The client sends a message and receives responses incrementally as the agent processes the request. This type of request is useful for long-running tasks or when you

want to update the user in real-time.

In both cases, requests usually include a `role` (for example, user) and the message content. More complex agents may return **task objects** instead of immediate messages, allowing for task tracking or cancellation.

Handle the agent response

Agent responses may include:

- **Direct messages:** Immediate outputs from the agent, such as text or structured content.
- **Task-based responses:** Objects representing ongoing tasks, which may require follow-up calls to check status or retrieve results.

Clients should be prepared to handle both response types and interpret the returned data appropriately.

Interacting with the agent

- Each request should be uniquely identifiable, often using a generated ID.
- Streaming responses are asynchronous and may provide partial results before the final output.
- Simple agents may return messages directly, while more advanced agents may manage multiple tasks simultaneously.

Connecting a client to your agent server involves **fetching the Agent Card**, **establishing a connection**, **sending requests**, and **handling responses**. By grasping these core concepts, you can confidently interact with your remote agent, whether you're sending simple messages or managing complex tasks.

Exercise - Connect to remote Azure AI Agents with the A2A protocol

Connect to remote agents with A2A protocol

In this exercise, you'll use Azure AI Agent Service with the A2A protocol to create simple remote agents that interact with one another. These agents will assist technical writers with preparing their developer blog posts. A title agent will generate a headline, and an outline agent will use the title to develop a concise outline for the article. Let's get started

Module assessment

1. What is the primary role of an A2A server? It routes requests between clients and connected agents.
2. What does the Agent Executor do in an A2A agent? Processes incoming requests and generates responses or events.
3. What is an agent card used for in A2A? It provides metadata about the agent, such as its capabilities and available functions.

Summary

In this module, you learned how to connect Python clients to Azure AI Agents using the **Agent-to-Agent (A2A) protocol**.

By running an A2A server and connecting your client, you explored **how agents are discovered and communicated with dynamically using the Agent Card**. You learned about **executors, which handle agent requests**, and **how messages—both streaming and non-streaming—flow between clients and agents**. Understanding these concepts allows you to build flexible, discoverable agent networks that can delegate tasks and respond to requests across distributed environments.

Analyze text with Azure AI Language

The Azure AI Language service enables you to create intelligent apps and services that **extract semantic information from text**.

Learning objectives

In this module, you'll learn how to use the Azure AI Language service to:

- **Detect language** from text
- Analyze text **sentiment**
- **Extract key phrases, entities, and linked entities**

Introduction

Every day, the world generates a vast quantity of data; much of it text-based in the form of emails, social media posts, online reviews, business documents, and more. Artificial intelligence techniques that apply statistical and semantic models enable you to create applications that extract meaning and insights from this text-based data.

The Azure AI Language provides an API for **common text analysis techniques** that you can easily integrate into your own application code.

In this module, you will learn how to use Azure AI Language to:

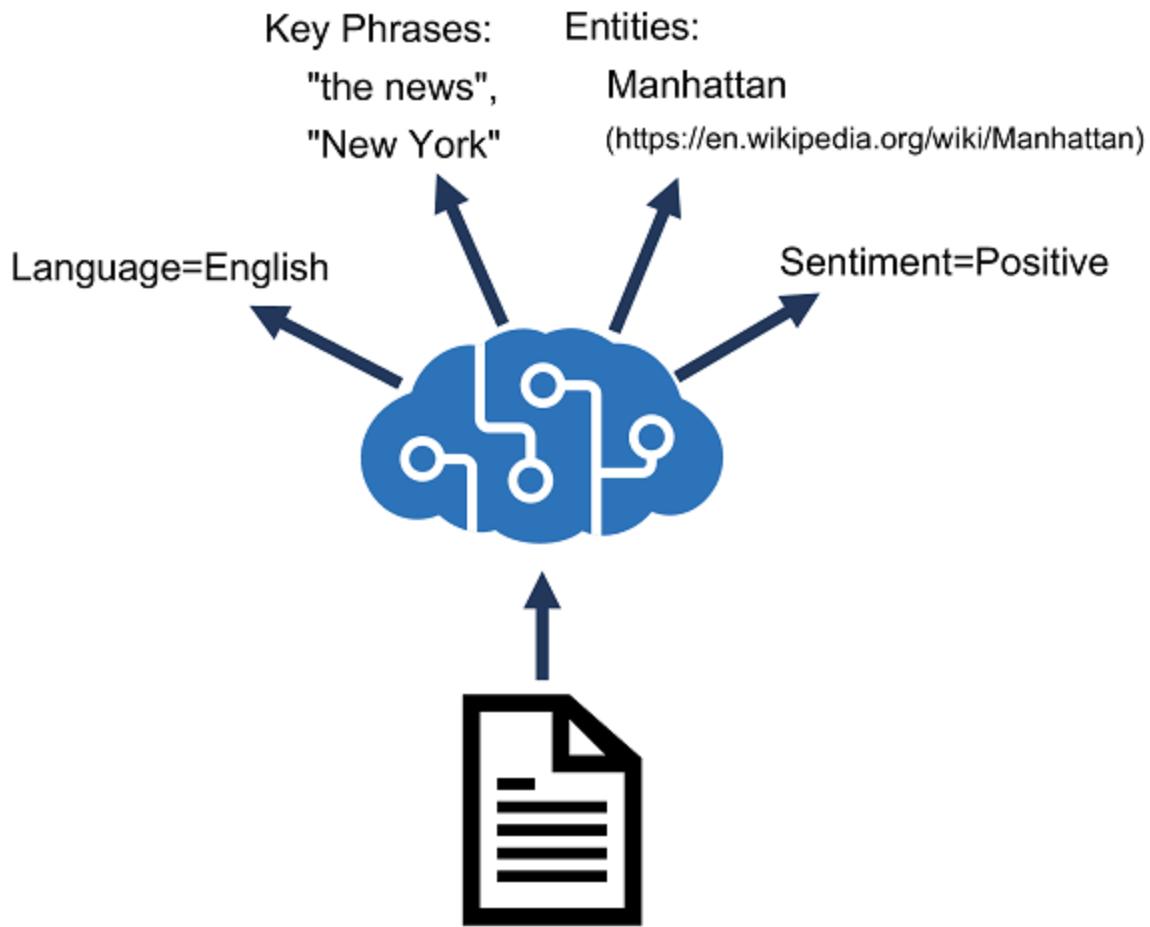
- Detect language from text.
- Analyze text sentiment.
- Extract key phrases, entities, and linked entities.

Provision an Azure AI Language resource

Azure AI Language is designed to help you extract information from text. It provides functionality that you can use for tasks like:

- **Language detection** - determining the language in which text is written.

- **Key phrase extraction** - identifying **important words and phrases** in the text that indicate the **main points**.
- **Sentiment analysis** - quantifying how positive or negative the text is.
- **Named entity recognition** - detecting references to entities, including **people, locations, time periods, organizations**, and more.
- **Entity linking** - identifying specific entities by providing **reference links to Wikipedia articles**.



Azure resources for text analysis

To use Azure AI Language to analyze text, you must provision a resource for it in your Azure subscription. You can **provision a resource through Azure AI Foundry portal**.

After you have provisioned a suitable resource in your Azure subscription, you can use its endpoint and one of its keys to call the Azure AI Language APIs from your code. You can call the Azure AI

Language APIs by **submitting requests in JSON format to the REST interface**, or by **using any of the available programming language-specific SDKs**.

Note: The code examples in the subsequent units in this module show the JSON requests and responses exchanged with the REST interface. When using an SDK, the JSON requests are abstracted by appropriate objects and methods that encapsulate the same data values. You'll get a chance to try the SDK for C# or Python for yourself in the exercise later in the module.

Detect language

The **Azure AI Language detection API** evaluates text input and, for each document submitted, returns **language identifiers** with a **score indicating the strength of the analysis**.

This capability is useful for content stores that collect arbitrary text, where language is unknown. Another scenario could involve a chat bot. If a user starts a session with the chat bot, **language detection can be used to determine which language they are using** and allow you to configure your bot responses in the appropriate language.

You can parse the results of this analysis to determine which language is used in the input document. The response also returns a score, which reflects **the confidence of the model (a value between 0 and 1)**.

Language detection can work with documents or single phrases. It's important to note that the **document size must be under 5,120 characters**. The **size limit is per document** and **each collection is restricted to 1,000 items (IDs)**. A sample of a properly formatted JSON payload that you might submit to the service in the request body is shown here, including a collection of documents, each containing a unique id and the text to be analyzed. Optionally, you can provide a countryHint to improve prediction performance.

```
{  
  "kind": "LanguageDetection",  
  "parameters": {  
    "modelVersion": "latest"  
  },  
  "analysisInput":{  
    "documents": [  
      {  
        "id": "1",  
        "text": "Hello world",  
        "countryHint": "US"  
      },  
      {  
        "id": "2",  
        "text": "Bonjour tout le monde"  
      }  
    ]  
  }  
}
```

The service will return a JSON response that contains a result for each document in the request body, **including the predicted language and a value indicating the confidence level of the prediction**. The confidence level is a value ranging from 0 to 1 with values closer to 1 being a higher confidence level. Here's an example of a standard JSON response that maps to the above request JSON.

```
{
  "kind": "LanguageDetectionResults",
  "results": [
    {
      "documents": [
        {
          "detectedLanguage": {
            "confidenceScore": 1,
            "iso6391Name": "en",
            "name": "English"
          },
          "id": "1",
          "warnings": []
        },
        {
          "detectedLanguage": {
            "confidenceScore": 1,
            "iso6391Name": "fr",
            "name": "French"
          },
          "id": "2",
          "warnings": []
        }
      ],
      "errors": [],
      "modelVersion": "2022-10-01"
    }
  ]
}
```

In our sample, all of the languages show a confidence of 1, mostly because the text is relatively simple and easy to identify the language for.

Multilingual Content

If you pass in a document that has multilingual content, the service will behave a bit differently. **Mixed language content within the same document returns the language with the largest representation in the content, but with a lower positive rating**, reflecting the marginal strength of that assessment. In the following example, the input is a blend of English, Spanish, and French. The analyzer uses statistical analysis of the text to determine the predominant language.

Textual Content not Parsable

The last condition to consider is when there is ambiguity as to the language content. The scenario might happen if you submit textual content that the analyzer is not able to parse, for example because of character encoding issues when converting the text to a string variable. As a result, the response for the language name and ISO code will indicate (unknown) and the score value will be returned as 0. The following example shows how the response would look.

Extract key phrases

Key phrase extraction is the process of **evaluating the text of a document, or documents, and then identifying the main points around the context of the document(s)**.

Key phrase extraction **works best for larger documents (the maximum size that can be analyzed is 5,120 characters)**.

As with language detection, the REST interface enables you to submit one or more documents for analysis.

Analyze sentiment

Sentiment analysis is used to **evaluate how positive or negative a text document is**, which can be useful in various workloads, such as:

- Evaluating a movie, book, or product by quantifying sentiment based on reviews.
- Prioritizing customer service responses to correspondence received through email or social media messaging.

When using Azure AI Language to evaluate sentiment, the response includes **overall document sentiment and individual sentence sentiment** for each document submitted to the service.

For example, you could submit a single document for sentiment analysis like this:

Sentence sentiment is based on confidence scores for positive, negative, and neutral classification values between 0 and 1.

Overall document sentiment is based on sentences:

- If all sentences are neutral, the overall sentiment is neutral.
- If sentence classifications include only positive and neutral, the overall sentiment is positive.
- If the sentence classifications include only negative and neutral, the overall sentiment is negative.
- If the sentence classifications include positive and negative, the overall sentiment is **mixed**.

Extract entities

Named Entity Recognition identifies entities that are mentioned in the text. Entities are grouped into **categories and subcategories**, for example:

- Person
- Location
- DateTime
- Organization
- Address
- Email
- URL

Note: For a full list of categories, see the [documentation](#).

Input for entity recognition is similar to input for other Azure AI Language API functions:

```
{  
  "kind": "EntityRecognition",  
  "parameters": {  
    "modelVersion": "latest"  
  },  
  "analysisInput": {  
    "documents": [  
      {  
        "id": "1",  
        "language": "en",  
        "text": "Joe went to London on Saturday"  
      }  
    ]  
  }  
}
```

The response includes a list of categorized entities found in each document:

```
{
  "kind": "EntityRecognitionResults",
  "results": {
    "documents": [
      {
        "entities": [
          {
            "text": "Joe",
            "category": "Person",
            "offset": 0,
            "length": 3,
            "confidenceScore": 0.62
          },
          {
            "text": "London",
            "category": "Location",
            "subcategory": "GPE",
            "offset": 12,
            "length": 6,
            "confidenceScore": 0.88
          },
          {
            "text": "Saturday",
            "category": "DateTime",
            "subcategory": "Date",
            "offset": 22,
            "length": 8,
            "confidenceScore": 0.8
          }
        ]
      },
      {
        "id": "1",
        "warnings": []
      }
    ],
    "errors": [],
    "modelVersion": "2021-01-15"
  }
}
```

To learn more about entities see the [Build a conversational language understanding model](#) module.

Extract linked entities

In some cases, the same name might be applicable to more than one entity. For example, does an instance of the word "Venus" refer to the planet or the goddess from mythology?

Entity linking can be used to **disambiguate entities of the same name by referencing an article in a knowledge base**. Wikipedia provides the knowledge base for the **Text Analytics service**. Specific article links are determined based on entity context within the text.

For example, "I saw Venus shining in the sky" is associated with the [Planet Venus](#); while "Venus, the goddess of beauty" is associated with [Godness Venus](#).

As with all Azure AI Language service functions, you can submit one or more documents for analysis:

```
{
  "kind": "EntityLinking",
  "parameters": {
    "modelVersion": "latest"
  },
  "analysisInput": {
    "documents": [
      {
        "id": "1",
        "language": "en",
        "text": "I saw Venus shining in the sky"
      }
    ]
  }
}
```

The response includes the entities identified in the text along with links to associated articles:

```
{  
  "kind": "EntityLinkingResults",  
  "results": {  
    "documents": [  
      {  
        "id": "1",  
        "entities": [  
          {  
            "bingId": "89253af3-5b63-e620-9227-f839138139f6",  
            "name": "Venus",  
            "matches": [  
              {  
                "text": "Venus",  
                "offset": 6,  
                "length": 5,  
                "confidenceScore": 0.01  
              }  
            ],  
            "language": "en",  
            "id": "Venus",  
            "url": "https://en.wikipedia.org/wiki/Venus",  
            "dataSource": "Wikipedia"  
          }  
        ],  
        "warnings": []  
      }  
    ],  
    "errors": [],  
    "modelVersion": "2021-06-01"  
  }  
}
```

Exercise - Analyze text

In this exercise, you use the Azure AI Language SDK to develop a client application that analyzes text.

Analyze Text

Azure Language supports analysis of text, including language detection, sentiment analysis, key phrase extraction, and entity recognition.

For example, suppose a travel agency wants to process hotel reviews that have been submitted to the company's web site. By using the Azure AI Language, they can determine the language each review is written in, the sentiment (positive, neutral, or negative) of the reviews, key phrases that might indicate the main topics discussed in the review, and named entities, such as places, landmarks, or people mentioned in the reviews.

Module assessment

1. How should you create an application that monitors the comments on your company's web site and flags any negative posts? **use the Azure AI service to perform sentiment analysis of the comments.**
2. You are analyzing text that contains the word "Paris". How might you determine if this word refers to the French city or the character in Homer's "The Iliad"? **Use the Azure AI Language Service to extract linked entities.**

Summary

In this module, you learned how to use Azure AI Language to:

- **Detect language** from text.
- Analyze text **sentiment**.
- Extract **key phrases, entities, and linked entities**.

To learn more about Azure AI Language and some of the concepts covered in this module, you can explore the following documentation pages:

- [Azure AI Language documentation](#)
- [Build a conversational language understanding model](#)
- [Create a custom named entity extraction solution](#)

Create question answering solutions with Azure AI Language

The question answering capability of the Azure AI Language service makes it easy to build applications in which **users ask questions using natural language and receive appropriate answers.**

Learning objectives

After completing this module, you will be able to:

- Understand question answering and how it compares to language understanding.
- Create, test, publish, and consume a **knowledge base**.
- Implement **multi-turn conversation** and **active learning**.
- Create a question answering bot to interact with using natural language.

Introduction

A common pattern for "intelligent" applications is to enable users to ask questions using natural language, and receive appropriate answers. In effect, this kind of solution brings **conversational intelligence to a traditional frequently asked questions (FAQ) publication**.

In this module, you will learn how to use Azure AI Language to **create a knowledge base of question and answer pairs** that can support an application or bot.

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

- Understand question answering and how it compares to language understanding.
- Create, test, publish and consume a knowledge base.
- Implement multi-turn conversation and active learning.
- Create a question answering bot to interact with using natural language.

Understand question answering

Azure AI Language includes a question answering capability, which enables you to define a knowledge base of question and answer pairs that can be queried using natural language input. The knowledge base can be published to a REST endpoint and consumed by client applications, commonly bots.

The knowledge base can be created from existing sources, including:

- **Web sites containing frequently asked question (FAQ) documentation.**
- Files containing structured text, such as brochures or user guides.
- Built-in chit chat question and answer pairs that encapsulate common conversational exchanges.

Note: The question answering capability of Azure AI Language is a newer version of the QnA Service, which still exists as a standalone service. To learn how to migrate a QnA Maker knowledge base to Azure AI Language, see the [migration guide](#).

Compare question answering to Azure AI Language understanding

A question answering knowledge base is a form of language model, which raises the question of when to use question answering, and when to use the conversational language understanding capabilities of Azure AI Language.

The two features are similar in that they both enable you to define a language model that can be queried using natural language expressions. However, there are some differences in the use cases that they are designed to address, as shown in the following table:

	Question answering	Language understanding
Usage pattern	User submits a question, expecting an answer	User submits an utterance, expecting an appropriate response or action
Query processing	Service uses natural language understanding to match the question to an answer in the knowledge base	Service uses natural language understanding to interpret the utterance, match it to an intent, and identify entities

	Question answering	Language understanding
Response	Response is a static answer to a known question	Response indicates the most likely intent and referenced entities
Client logic	Client application typically presents the answer to the user	Client application is responsible for performing appropriate action based on the detected intent

The two services are in fact complementary. You can build **comprehensive natural language solutions that combine language understanding models and question answering knowledge bases.**

Create a knowledge base

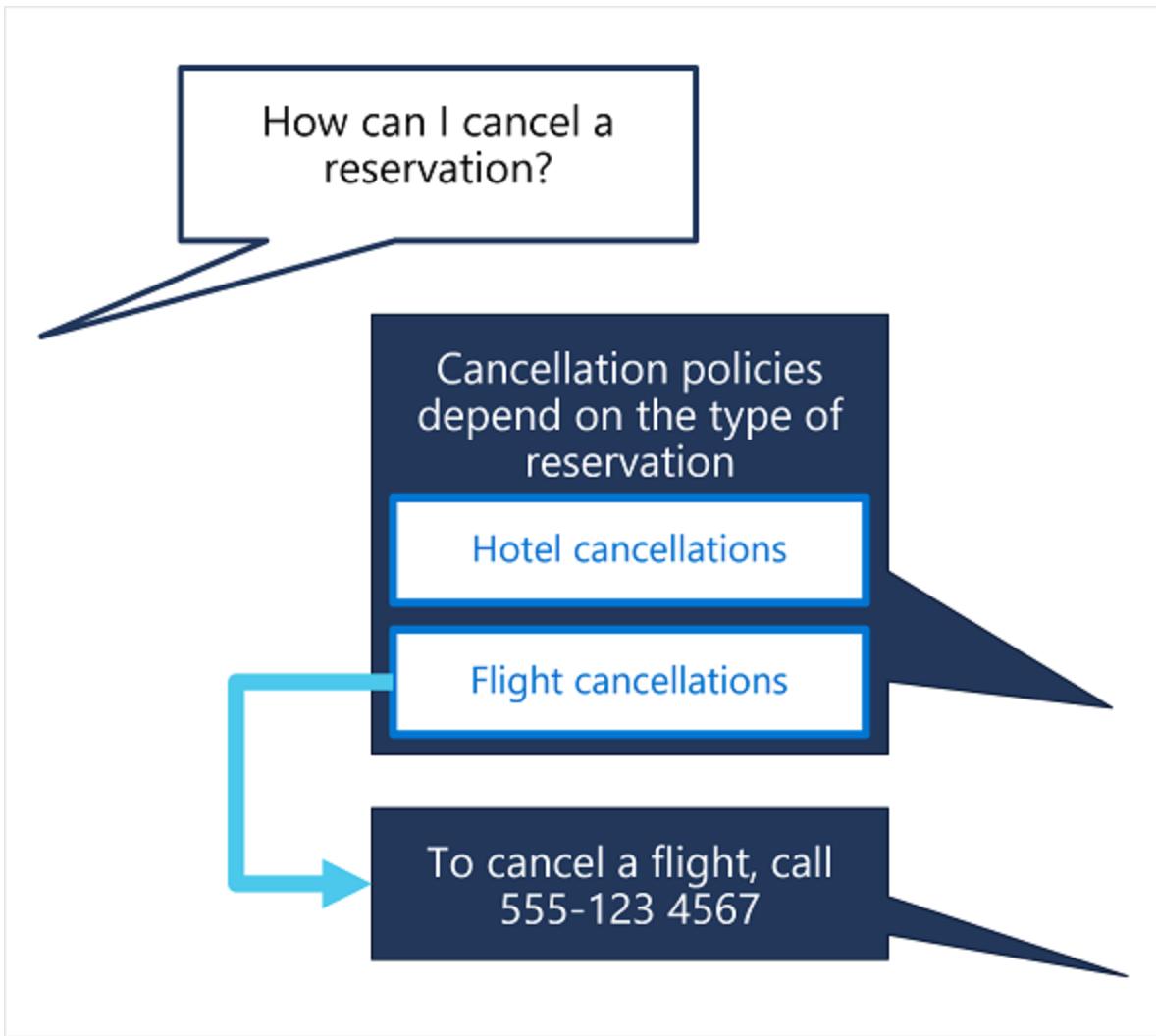
To create a question answering solution, you can use the REST API or SDK to write code that defines, trains, and publishes the knowledge base. However, it's more common to use the [Language Studio](#) web interface to define and manage a knowledge base.

To create a knowledge base you:

1. Sign in to Azure portal.
2. Search for Azure AI services using the search field at the top of the portal.
3. Select Create under the Language Service resource.
4. Create a resource in your Azure subscription:
 - Enable the question answering feature.
 - Create or select an Azure AI Search resource to host the knowledge base index.
5. In Language Studio, select your Azure AI Language resource and create a Custom question answering project.
6. Add one or more data sources to populate the knowledge base:
 - URLs for web pages containing FAQs.
 - Files containing structured text from which questions and answers can be derived.
 - Predefined chit-chat datasets that include common conversational questions and responses in a specified style.
7. Edit question and answer pairs in the portal.

Implement multi-turn conversation

Although you can often create an effective knowledge base that consists of individual question and answer pairs, sometimes you might need to **ask follow-up questions to elicit more information from a user before presenting a definitive answer**. This kind of interaction is referred to as a **multi-turn conversation**.



You can **enable multi-turn responses when importing questions and answers from an existing web page or document based on its structure**, or you **can explicitly define follow-up prompts and responses for existing question and answer pairs**.

For example, suppose an initial question for a travel booking knowledge base is "How can I cancel a reservation?". A reservation might refer to a hotel or a flight, so a follow-up prompt is required to clarify this detail. The answer might consist of text such as "Cancellation policies depend on the type of reservation" and include follow-up prompts with links to answers about canceling flights and canceling hotels.

When you define a follow-up prompt for multi-turn conversation, you can link to an existing answer in the knowledge base or define a new answer specifically for the follow-up. You can also restrict the linked answer so that it is only ever displayed in the context of the multi-turn conversation initiated by the original question.

Test and publish a knowledge base

After you have defined a knowledge base, you can train its natural language model, and test it before publishing it for use in an application or bot.

Testing a knowledge base

You can test your knowledge base interactively in Language Studio, submitting questions and reviewing the answers that are returned. You can inspect the results to view their confidence scores as well as other potential answers.

The screenshot shows the Azure AI Language Studio interface. On the left, there's a sidebar with navigation links: Language Studio, Custom question answering, Azure Search, LearnFAQ, Manage sources, Edit knowledge base (which is selected and highlighted in grey), Deploy knowledge base, Review suggestions, and Project settings. The main area is titled 'Edit knowledge base' and shows 'Question answer pairs (150)'. It includes a search bar and a table of question-answer pairs. The table rows include:

Question	Answer	Confidence Score
I'm so bored of you	Swing and a miss.	112
Are you a woman?	That doesn't really apply to me.	68
Who's older?	I don't really have an age.	47
Will you ask me anything about myself	I'm a much better answerer than asker.	127
Do you have fingers?	I don't have the hardware for that.	74
Do you have a boss?	I'm here for you!	59
Can you jump rope?		209

To the right, a 'Test' pane is open, showing a conversation history and a message input field. The history includes:

- Hi!
- What is Microsoft Learn?

The message input field at the bottom says 'Type your message and press enter'.

Deploying a knowledge base

When you're happy with the performance of your knowledge base, you can **deploy it to a REST endpoint** that client applications can use to submit questions and receive answers. You can deploy it directly from [Language Studio](#).

Use a knowledge base

To consume the published knowledge base, you can use the REST interface.

The minimal request body for the function contains a question, like this:

```
{  
  "question": "What do I need to do to cancel a reservation?",  
  "top": 2,  
  "scoreThreshold": 20,  
  "strictFilters": [  
    {  
      "name": "category",  
      "value": "api"  
    }  
  ]  
}
```

Property	Description
question	Question to send to the knowledge base.
top	Maximum number of answers to be returned.
scoreThreshold	<i>Score threshold for answers returned.</i>
strictFilters	Limit to only answers that contain the specified metadata.

The response includes the closest question match that was found in the knowledge base, along with the **associated answer**, the **confidence score**, and other metadata about the question and answer pair:

```
{  
  "answers": [  
    {  
      "score": 27.74823341616769,  
      "id": 20,  
      "answer": "Call us on 555 123 4567 to cancel a reservation.",  
      "questions": [  
        "How can I cancel a reservation?"  
      ],  
      "metadata": [  
        {  
          "name": "category",  
          "value": "api"  
        }  
      ]  
    }  
  ]  
}
```

Improve question answering performance

After creating and testing a knowledge base, you can **improve its performance with active learning and by defining synonyms**.

Use active learning

Active learning can help you make continuous improvements to get better at answering user questions correctly over time. **People often ask questions that are phrased differently, but ultimately have the same meaning.** Active learning can help in situations like this because it enables you to consider alternate questions to each question and answer pair. Active learning is enabled by default.

To use active learning, you can do the following:

Create your question and answer pairs

You create pairs of questions and answers in Language Studio for your project. You can also import a file that contains question and answer pairs to upload in bulk.

Home Question answer pairs (0) Synonyms (0)

Question answer pairs (0) Synonyms (0)

+ X

Search pairs

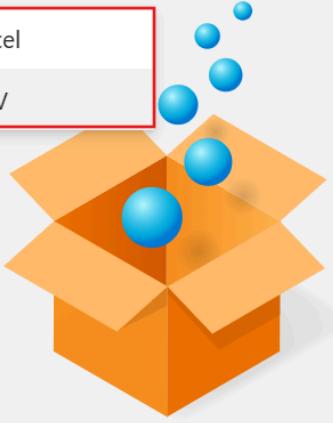
No QnA pairs. Select add to create a new QnA pair

Import questions and answers > Import as Excel
Export questions and answers > Import as TSV

Edit knowledge base

0 unstructured sources and 0 structured sources.
[View sources](#)

You haven't created any question and answer pairs yet.



Review suggestions

Active learning then begins to offer alternate questions for each question in your question and answer pairs. You access this from the Review suggestions pane:



Review suggestions



Review, accept, or reject suggested alternate phrases for questions. These suggestions come from users interacting with your bot. Only the questions with suggestions are shown.



[Accept all suggestions](#)

[Reject all suggestions](#)

[Show columns](#) ▾

3 pairs [Filter](#)



Questions ▾



Fix problems with Surface Pen



help with my surface pen



Try these apps with your pen



apps for surface pen



Write and draw



draw with your pen



can i draw with surface pen



how to write with surface pen



Answer and prompts ▾

Here are some things to try first if your Surface Pen won't write, open apps, or connect to Bluetooth.

[Check compatibility](#)

[Check Pen Settings](#)

Ready to write your ideas, take notes, and be more productive with ink? Get started with these apps.

[Microsoft Whiteboard](#)

[OneNote](#)

[Office](#)

[Microsoft To-Do](#)

Use your Surface Pen in any app that supports inking. To see which apps to start with, go to the section [Try these apps with your Pen](#bkmk_trytheseapps).

[Start inking with your ...](#)

[Enter text with your pen](#)

You review, and then accept or reject these alternate phrases suggested for each question by selecting the checkmark or delete symbol next to the alternate phrase. You can bulk accept or reject suggestions using the Accept all suggestions or Reject all suggestions option at the top.

You can also manually add alternate questions when you select Add alternate question for a pair in the Edit knowledge base pane:



Language Studio > Custom question answering > sample-project - Edit knowledge base



Question answer pairs (21) [Synonyms \(0\)](#)



[+](#) [B](#) [D](#) [V](#) [C](#) [X](#)



[Search pairs](#)



Replace Pen Tips



Here's how you can replace the pen tips:
1. Use something to pull and slide out...



Check Pen Settings



Check your pen settings To check your cursor, handwriting, and top button...



Check your Surface Pen Compatibility



Check if your Surface Pen is compatible with your Surface. For more info, go...



Fix problems with Surface Pen



Here are some things to try first if your

Edit knowledge base

0 unstructured sources and 2 structured sources. [View sources](#)

[Enable rich text](#) [Show context tree](#)

Replace Pen Tips

Source: Editorial [Edit](#)

Answer

Edit answer

Here's how you can replace the pen tips:

1. Use something to pull and slide out the pen tip, like tweezers. Pull until the entire tip is removed from your Surface Pen.
2. Insert the thinnest part of the replacement pen tip into your Surface Pen. Push until the tip locks into place.

Alternate questions (1)

Add an alternate question when there are multiple ways the same question can be asked. Alternate questions should be as semantically dissimilar as possible and should not exceed 10 alternate questions.

[+ Add alternate question](#)

Replace Pen Tips

Follow up prompts (0)

Metadata (0)

Note: To learn more about active learning, see [Enrich your project with active learning](#).

Define synonyms

Synonyms are useful when questions submitted by users might **include multiple different words to mean the same thing**. For example, a travel agency customer might refer to a "reservation" or a "booking". By defining these as synonyms, the question answering service can find an appropriate answer regardless of which term an individual customer uses.

To define synonyms, you **use the REST API to submit synonyms in the following JSON format**:

```
{  
  "synonyms": [  
    {  
      "alterations": [  
        "reservation",  
        "booking"  
      ]  
    }  
  ]  
}
```

Note: To learn more about synonyms, see the [Improve quality of response with synonyms](#).

Exercise - Create a question answering solution

Create a Question Answering Solution

One of the most common conversational scenarios is providing support through a knowledge base of frequently asked questions (FAQs). Many organizations publish FAQs as documents or web pages, which works well for a small set of question and answer pairs, but large documents can be difficult and time-consuming to search.

Azure AI Language includes a question answering capability that enables you to create a knowledge base of question and answer pairs that can be queried using natural language input, and is most commonly used as a resource that a bot can use to look up answers to questions submitted by users.

Module assessment

1. You want to create a knowledge base from an existing FAQ document. What should you do?
Create a new knowledge base, importing the existing FAQ document.
2. How can you add a multi-turn context for a question in an existing knowledge base? Add a follow-up prompt to the question.
3. How can you enable users to use your knowledge base through email? Create a bot based on your knowledge base and configure an email channel.

Summary

In this module, you have learned how to use the question answering capability of Azure AI Language to create a knowledge base of question and answer pairs that can support an application or bot.

Now that you've completed this module, you can:

- Understand **question answering** and how it compares to **language understanding**.
- Create, test, publish and consume a **knowledge base**.
- Implement **multi-turn conversation** and **active learning**.
- Create a **question answering bot** to interact with using natural language.

To learn more about the question answering capability of Azure AI Language, see the [Question answering documentation](#).

Build a conversational language understanding model

The Azure AI Language **conversational language understanding service (CLU)** enables you to train a model that apps can use to extract meaning from natural language.

Learning objectives

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

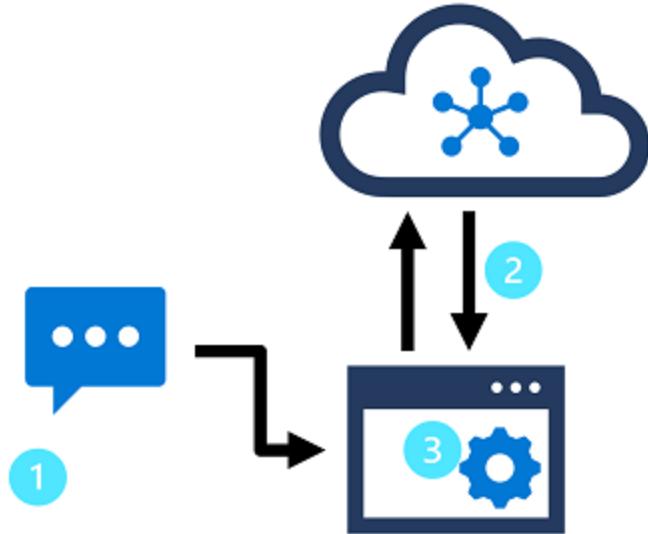
- Provision Azure resources for Azure AI Language resource
- Define intents, utterances, and entities
- Use patterns to differentiate similar utterances
- Use pre-built entity components
- Train, test, publish, and review an Azure AI Language model

Introduction

[Video](#)

Natural language processing (NLP) is a common AI problem in which software must be able to work with text or speech in the natural language form that a human user would write or speak. Within the broader area of NLP, **natural language understanding (NLU)** deals with the problem of determining semantic meaning from natural language - usually by using a trained language model.

A common design pattern for a natural language understanding solution looks like this:



In this design pattern:

1. An app accepts natural language input from a user.
2. A language model is used to determine semantic meaning (the user's intent).
3. The app performs an appropriate action.

Azure AI Language enables developers to build apps based on language models that can be trained with a relatively small number of samples to discern a user's intended meaning.

In this module, you'll learn how to use the service to create **a natural language understanding app** using Azure AI Language.

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

- Provision an Azure AI Language resource.
- Define intents, entities, and utterances.
- Use patterns to differentiate similar utterances.
- Use pre-built entity components.
- Train, test, publish, and review a model.

Understand prebuilt capabilities of the Azure AI Language service

The Azure AI Language service provides various features for understanding human language. You can use each feature to better communicate with users, better understand incoming communication, or use them together to provide more insight into what the user is saying, intending, and asking about.

Azure AI Language service features fall into two categories: **Pre-configured features**, and **Learned features**. Learned features require building and training a model to correctly predict appropriate labels, which is covered in upcoming units of this module.

This unit covers most of the capabilities of the Azure AI Language service, but head over to the [Azure AI Language service documentation](#) for a full list, including quickstarts and a full explanation of everything available.

Using these features in your app requires sending your query to the appropriate endpoint. The endpoint used to query a specific feature varies, but all of them are prefixed with the Azure AI Language resource you created in your Azure account, either when building your REST request or defining your client using an SDK. Examples of each can be found in the next unit.

Pre-configured features

The Azure AI Language service provides certain features without any model labeling or training. Once you create your resource, you can send your data and use the returned results within your app.

The following features are all pre-configured.

Summarization

Summarization is available for both documents and conversations, and will summarize the text into key sentences that are predicted to encapsulate the input's meaning.

Named entity recognition

Named entity recognition can **extract and identify entities, such as people, places, or companies**, allowing your app to recognize different types of entities for improved natural language responses. For example, given the text "The waterfront pier is my favorite Seattle attraction", Seattle would be identified and categorized as a location.

Personally identifiable information (PII) detection

PII detection allows you to **identify, categorize, and redact information that could be considered sensitive**, such as email addresses, home addresses, IP addresses, names, and protected health information. For example, if the text "email@contoso.com" was included in the query, the entire email address can be identified and redacted.

Key phrase extraction

Key phrase extraction is a feature that quickly pulls the main concepts out of the provided text. For example, given the text "Text Analytics is one of the features in Azure AI Services.", the service would extract "Azure AI Services" and "Text Analytics".

Sentiment analysis

Sentiment analysis identifies how **positive or negative** a string or document is. For example, given the text "Great hotel. Close to plenty of food and attractions we could walk to", the service would identify that as *positive* with a relatively high confidence score.

Language detection

Language detection takes one or more documents, and identifies the language for each. For example, if the text of one of the documents was "Bonjour", the service would identify that as *French*.

Learned features

Learned features require you to label data, train, and deploy your model to make it available to use in your application. These features allow you to customize what information is predicted or extracted.

Note: Quality of data greatly impacts the model's accuracy. Be intentional about what data is used, how well it is tagged or labeled, and how varied the training data is. For details, see recommendations for labeling data, which includes valuable guidelines for tagging data. Also see the evaluation metrics that can assist in learning where your model needs improvement.

Conversational language understanding (CLU)

CLU is one of the core custom features offered by Azure AI Language. CLU helps users to build **custom natural language understanding models to predict overall intent and extract important information from incoming utterances**. CLU does require data to be tagged by the user to teach it how to predict intents and entities accurately.

The exercise in this module will be building a CLU model and using it in your app.

Custom named entity recognition

Custom entity recognition takes custom labeled data and extracts specified entities from unstructured text. For example, if you have various contract documents that you want to extract involved parties from, you can train a model to recognize how to predict them.

Custom text classification

Custom text classification enables users to classify text or documents as custom defined groups. For example, you can train a model to look at news articles and identify the category they should fall into, such as News or Entertainment.

Question answering

Question answering is a mostly pre-configured feature that provides answers to questions provided as input. The data to answer these questions comes from documents like FAQs or manuals.

For example, say you want to make a virtual chat assistant on your company website to answer common questions. You could use a company FAQ as the input document to create the question and answer pairs. Once deployed, your chat assistant can pass input questions to the service, and get the answers as a result.

For a complete list of capabilities and how to use them, see the [Azure AI Language documentation](#).

Understand resources for building a conversational language understanding model

To use the Language Understanding service to develop a NLP solution, you'll need to [create a Language resource in Azure](#). That resource will be used for both authoring your model and processing prediction requests from client applications.

Tip: This module's lab covers building a model for conversational language understanding. For more focused modules on custom text classification and custom named entity recognition, see the custom solution modules in the [Develop natural language solutions](#) learning path.

Build your model

For features that require a model for prediction, you'll need to build, train and deploy that model before using it to make a prediction. This building and training will teach the Azure AI Language service what to look for.

First, you'll need to create your Azure AI Language resource in the [Azure portal](#). Then:

1. Search for **Azure AI services**.
2. Find and select **Language Service**.
3. Select *Create* under the **Language Service**.
4. Fill out the necessary details, choosing the region closest to you geographically (for best performance) and giving it a unique name.

Once that resource has been created, you'll need **a key and the endpoint**. You can find that on the left side under Keys and Endpoint of the resource overview page.

Use Language Studio

For a more visual method of building, training, and deploying your model, you can use [Language Studio](#) to achieve each of these steps. On the main page, you can choose to create a Conversational language understanding project. Once the project is created, then go through the same process as above to build, train, and deploy your model.

Language Studio



Get started with Azure Cognitive Services for Language

Use our Natural Language Processing (NLP) features to analyze your textual data using state-of-the-art pre-configured AI models or customize your own models to fit your scenario.



Featured

Extract information

Classify text

Understand questions and conversational language

Summarize text

Translate text

Retrieve the most appropriate answer to questions using question answering (CQA) or classify intents and extract entities for conversational utterances using conversational language understanding (CLU). Use orchestration workflow to create one project that routes queries between multiple CQA and CLU projects. [Learn more about understanding conversational language](#).



Answer questions

Use the prebuilt question answering API to get answers to questions over unstructured text.

[Try it out](#)

Custom question answering

Next generation of QnAMaker

Customize the list of questions and answers extracted from your content corpus to provide a conversational experience that suits your needs.

[Open Custom question answering](#)

Conversational language understanding

Next generation of LUIS

Classify utterances into intents and extract information with entities to build natural language into apps, bots, and IoT devices.
[Open Conversational language understanding](#)

The lab in this module will walk through using Language Studio to build your model. If you'd like to learn more, see the [Language Studio quickstart](#).

Use the REST API

One way to build your model is through the REST API. The pattern would be to create your project, import data, train, deploy, then use your model.

These tasks are done **asynchronously**; you'll need to submit a request to the appropriate URI for each step, and then **send another request to get the status of that job**.

For example, if you want to deploy a model for a conversational language understanding project, you'd submit the deployment job, and then check on the deployment job status.

Authentication

For each call to your Azure AI Language resource, you authenticate the request by providing the following header.

Key	Value
Ocp-Apim-Subscription-Key	The key to your resource

Request deployment

Submit a POST request to the following endpoint.

{ENDPOINT}/language/authoring/analyze-conversations/projects/{PROJECT-NAME}/deployments/{DEPLOYMENT-NAME}

Placeholder	Value	Example
{ENDPOINT}	The endpoint of your Azure AI Language resource	https://<your-subdomain>.cognitiveservices.azure.com
{PROJECT-NAME}	The name for your project. This value is case-sensitive	myProject
{DEPLOYMENT-NAME}	The name for your deployment. This value is case-sensitive	staging
{API-VERSION}	The version of the API you're calling	2022-05-01

Include the following `body` with your request.

```
{
  "trainedModelLabel": "{MODEL-NAME}",
}
```

Placeholder	Value
MODEL-NAME	The model name that will be assigned to your deployment. This value is case-sensitive.

Successfully submitting your request will receive a `202` response, with a response header of `operation-location`. This header will have a URL with which to request the status, formatted like this:

```
{ENDPOINT}/language/authoring/analyze-conversations/projects/{PROJECT-NAME}/deployments/{DEPLOYMENT-NAME}
```

Get deployment status

Submit a `GET` request to the URL from the response header above. The values will already be filled out based on the initial deployment request.

```
{ENDPOINT}/language/authoring/analyze-conversations/projects/{PROJECT-NAME}/deployments/{DEPLOYMENT-NAME}
```

Placeholder	Value
ENDPOINT	The endpoint for authenticating your API request
PROJECT-NAME	The name for your project (case-sensitive)
DEPLOYMENT-NAME	The name for your deployment (case-sensitive)
JOB-ID	The ID for locating your model's training status, found in the header value detailed above in the deployment request
API-VERSION	The version of the API you're calling

The response body will give the deployment `status` details. The `status` field will have the value of `succeeded` when the deployment is complete.

```
{
  "jobId": "{JOB-ID}",
  "createdDateTime": "String",
  "lastUpdatedDateTime": "String",
  "expirationDateTime": "String",
  "status": "running"
}
```

For a full walkthrough of each step with example requests, see the [conversational understanding quickstart](#).

Query your model

To query your model for a prediction, you can [use SDKs in C# or Python](#), or [use the REST API](#).

Query using SDKs

To query your model using an SDK, you **first need to create your client**. Once you have your client, you **then use it to call the appropriate endpoint**.

```
language_client = TextAnalyticsClient(  
    endpoint=endpoint,  
    credential=credentials)  
response = language_client.extract_key_phrases(documents = documents)[0]
```

Other language features, such as the conversational language understanding, require the request be built and sent differently.

```
result = client.analyze_conversation(  
    task={  
        "kind": "Conversation",  
        "analysisInput": {  
            "conversationItem": {  
                "participantId": "1",  
                "id": "1",  
                "modality": "text",  
                "language": "en",  
                "text": query  
            },  
            "isLoggingEnabled": False  
        },  
        "parameters": {  
            "projectName": cls_project,  
            "deploymentName": deployment_slot,  
            "verbose": True  
        }  
    }  
)
```

Query using the REST API

To query your model using REST, create a `POST` request to the appropriate URL with the appropriate body specified. For built in features such as language detection or sentiment analysis, you'll query the `analyze-text` **endpoint**.

Tip: Remember each request needs to be authenticated with your Azure AI Language resource key in the `Ocp-Apim-Subscription-Key` header

```
{ENDPOINT}/language/:analyze-text?api-version={API-VERSION}
```

Placeholder	Value
ENDPOINT	The endpoint for authenticating your API request
API-VERSION	The version of the API you're calling

Within the body of that request, you must specify the `kind` parameter, which tells the service what type of language understanding you're requesting.

If you want to detect the language, for example, the JSON body would look something like the following.

```
{
  "kind": "LanguageDetection",
  "parameters": {
    "modelVersion": "latest"
  },
  "analysisInput":{
    "documents":[
      {
        "id":"1",
        "text": "This is a document written in English."
      }
    ]
  }
}
```

Other language features, such as the conversational language understanding, require the request be routed to a different endpoint. For example, the conversational language understanding request would be sent to the following.

{ENDPOINT}/language/:analyze-conversations?api-version={API-VERSION}

Placeholder	Value
ENDPOINT	The endpoint for authenticating your API request
API-VERSION	The version of the API you're calling

That request would include a JSON body similar to the following.

```
{
  "kind": "Conversation",
  "analysisInput": {
    "conversationItem": {
      "id": "1",
      "participantId": "1",
      "text": "Sample text"
    }
  },
  "parameters": {
    "projectName": "{PROJECT-NAME}",
    "deploymentName": "{DEPLOYMENT-NAME}",
    "stringIndexType": "TextElement_V8"
  }
}
```

Placeholder	Value
PROJECT-NAME	The name of the project where you built your model
DEPLOYMENT-NAME	The name of your deployment

Sample response

The query response from an SDK will be an object returned, which varies depending on the feature (such as in `response.key_phrases` or `response.value`). The REST API will return JSON that would be similar to the following.

```
{  
  "kind": "KeyPhraseExtractionResults",  
  "results": {  
    "documents": [{  
      "id": "1",  
      "keyPhrases": ["modern medical office", "Dr. Smith", "great staff"],  
      "warnings": []  
    }],  
    "errors": [],  
    "modelVersion": "{VERSION}"  
  }  
}
```

For other models like conversational language understanding, a sample response to your query would be similar to the following.

```
{  
  "kind": "ConversationResult",  
  "result": {  
    "query": "String",  
    "prediction": {  
      "topIntent": "intent1",  
      "projectKind": "Conversation",  
      "intents": [  
        {  
          "category": "intent1",  
          "confidenceScore": 1  
        },  
        {  
          "category": "intent2",  
          "confidenceScore": 0  
        }  
      ],  
      "entities": [  
        {  
          "category": "entity1",  
          "text": "text",  
          "offset": 7,  
          "length": 4,  
          "confidenceScore": 1  
        }  
      ]  
    }  
  }  
}
```

The SDKs for both Python and C# return JSON that is very similar to the REST response.

For full documentation on features, including examples and how-to guides, see the [Azure AI Language documentation](#) documentation pages.

Define intents, utterances, and entities

Utterances are the phrases that a user might enter when interacting with an application that uses your language model. An **intent** represents **a task or action the user wants to perform**, or more simply

the **meaning of an utterance**. You create a model by defining **intents** and associating them with one or more **utterances**.

For example, consider the following list of intents and associated utterances:

- GetTime:
 - "What time is it?"
 - "What is the time?"
 - "Tell me the time"
- GetWeather:
 - "What is the weather forecast?"
 - "Do I need an umbrella?"
 - "Will it snow?"
- TurnOnDevice
 - "Turn the light on."
 - "Switch on the light."
 - "Turn on the fan"
- None:
 - "Hello"
 - "Goodbye"

In your model, you must define the **intents** that you want your model to understand, so spend some time considering the **domain** your model must support and the kinds of **actions or information** that users might request. In addition to the intents that you define, every model includes a `None` intent that you should use to explicitly identify utterances that a user might submit, but for which there is *no specific action required* (for example, conversational greetings like "hello") or that *fall outside of the scope of the domain for this model*.

After you've identified the intents your model must support, it's important to capture various different example **utterances** for each intent. Collect utterances that you think users will enter; including utterances meaning the same thing but that are constructed in different ways. Keep these guidelines in mind:

- Capture multiple different examples, or alternative ways of saying the same thing
- Vary the length of the utterances from short, to medium, to long
- Vary the location of the noun or subject of the utterance. Place it at the beginning, the end, or somewhere in between
- Use correct grammar and incorrect grammar in different utterances to offer good training data examples

- The precision, consistency and completeness of your labeled data are key factors to determining model performance.
 - Label **precisely**: Label each entity to its right type always. Only include what you want extracted, avoid unnecessary data in your labels.
 - Label **consistently**: The same entity should have the same label across all the utterances.
 - Label **completely**: Label all the instances of the entity in all your utterances.

Entities are used to add specific context to intents. For example, you might define a `TurnOnDevice` intent that can be applied to multiple devices, and use entities to define the different devices.

Consider the following utterances, intents, and entities:

Utterance	Intent	Entities
What is the time?	GetTime	
What time is it in <i>London</i> ?	GetTime	Location (<i>London</i>)
What's the weather forecast for <i>Paris</i> ?	GetWeather	Location (<i>Paris</i>)
Will I need an umbrella <i>tonight</i> ?	GetWeather	Time (<i>tonight</i>)
What's the forecast for <i>Seattle</i> <i>tomorrow</i> ?	GetWeather	Location (<i>Seattle</i>), Time (<i>tomorrow</i>)
Turn the <i>light</i> on.	TurnOnDevice	Device (<i>light</i>)
Switch on the <i>fan</i> .	TurnOnDevice	Device (<i>fan</i>)

You can **split entities into a few different component types**:

- **Learned** entities are **the most flexible kind of entity, and should be used in most cases**. You define a learned component with a suitable name, and then associate words or phrases with it in training utterances. When you train your model, it learns to match the appropriate elements in the utterances with the entity.
- **List** entities are useful when you need an entity with a specific set of possible values - for example, days of the week. You can include synonyms in a list entity definition, so you could define a `DayOfWeek` entity that includes the values "Sunday", "Monday", "Tuesday", and so on; each with synonyms like "Sun", "Mon", "Tue", and so on.
- **Prebuilt** entities are useful for common types such as **numbers, datetimes, and names**. For example, when prebuilt components are added, you will automatically detect values such as "6" or organizations such as "Microsoft". You can see this article for a list of [supported prebuilt entities](#).

Use patterns to differentiate similar utterances

In some cases, a model might contain **multiple intents** for which utterances are likely to be similar. You can use the pattern of utterances to disambiguate the intents while minimizing the number of sample utterances.

For example, consider the following utterances:

- "Turn on the kitchen light"
- "Is the kitchen light on?"
- "Turn off the kitchen light"

These utterances are syntactically similar, with only a few differences in words or punctuation. However, they represent three different intents (which could be named `TurnOnDevice`, `GetDeviceStatus`, and `TurnOffDevice`). Additionally, the intents could apply to a wide range of entity values. In addition to "kitchen light", the intent could apply to "living room light", "television", or any other device that the model might need to support.

To correctly train your model, provide a handful of examples of each intent that specify the different formats of utterances.

- TurnOnDevice:
 - "Turn on the {DeviceName}"
 - "Switch on the {DeviceName}"
 - "Turn the {DeviceName} on"
- GetDeviceStatus:
 - "Is the {DeviceName} on[?]"
- TurnOffDevice:
 - "Turn the {DeviceName} off"
 - "Switch off the {DeviceName}"
 - "Turn off the {DeviceName}"

When you teach your model with each different type of utterance, the Azure AI Language service can learn how to categorize intents correctly based off format and punctuation.

Use pre-built entity components

You can **create your own language models by defining all the intents and utterances it requires**, but often you can use **prebuilt components** to **detect common entities such as numbers, emails, URLs, or choices**.

For a full list of prebuilt entities the Azure AI Language service can detect, see the list of [supported prebuilt entity components](#).

Using prebuilt components allows you to let the Azure AI Language service automatically detect the specified type of entity, and not have to train your model with examples of that entity.

To add a prebuilt component, you can create an entity in your project, then select **Add new prebuilt** to that entity to detect certain entities.

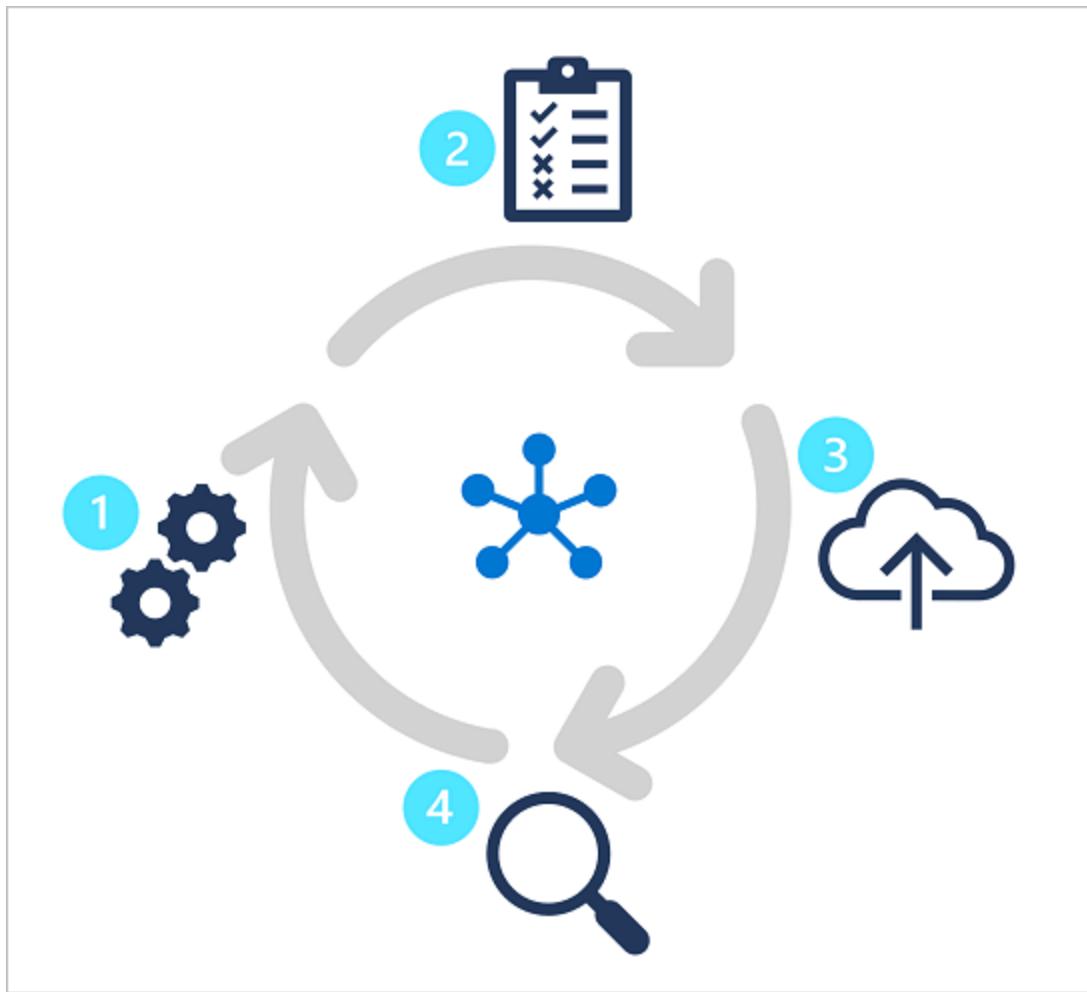
The screenshot shows the Azure AI Language Studio interface with the following details:

- Header:** Azure AI | Language Studio
- Breadcrumbs:** Language Studio > Conversational Language Understanding projects > Clock - Schema definition > Location - Entity components
- Left Sidebar:** Icons for Home, Projects, Entities, Utterances, Labels, Components, and Help.
- Title:** Location
- Description:** Each entity is made of multiple components that you can define. You can define one or more components. If you have more than one component, the logic that combines their responses is defined in the Overlap method tab.
- Entity components Tab:** Selected. Options: Entity components (highlighted), Entity Options.
- Actions:** Edit, Delete.
- Component Types:**
 - Learned:** Required component (switch is on).
 - Prebuilt (0):** Not Required (switch is off). Description: A prebuilt component gets your entity to extract common types such as numbers, dates, times, and others. A red box highlights the "Prebuilt name" dropdown menu, which has "Prebuilt name" and "Add new prebuilt" options.
 - Regular expression (0):** Not Required (switch is off). Description: A regex component matches regular expressions for common patterns. You can associate a key to each expression. A red box highlights the "Regex key" dropdown menu, which has "Regex key ↑" and "Add expression" options.
- Message:** No items found.

You can have **up to five prebuilt components per entity**. Using prebuilt model elements can significantly reduce the time it takes to develop a conversational language understanding solution.

Train, test, publish, and review a conversational language understanding model

Creating a model is an iterative process with the following activities:



- **Train a model to learn intents and entities from sample utterances.**
- **Test the model interactively** or using a testing dataset with known labels
- **Deploy a trained model** to a *public endpoint* so client apps can use it
- **Review predictions and iterate on utterances** to train your model

By following this iterative approach, you can improve the language model over time based on user input, helping you develop solutions that reflect the way users indicate their intents using natural language.

Exercise - Build an Azure AI services conversational language understanding model

In this exercise, you use Azure AI Language to build a conversational language understanding model.

Create a language understanding model with the Language service

The Azure AI Language service enables you to define a **conversational language understanding model** that applications can use to interpret natural language utterances from users (text or spoken input), predict the users intent (what they want to achieve), and identify any entities to which the intent should be applied.

For example, a conversational language model for a clock application might be expected to process input such as:

What is the time in London?

This kind of input is an example of an **utterance** (something a user might say or type), for which the desired intent is to get the time in a specific location (an entity); in this case, *London*.

NOTE: The task of a conversational language model is to predict the user's intent and identify any entities to which the intent applies. It is not the job of a conversational language model to actually perform the actions required to satisfy the intent. For example, a clock application can use a conversational language model to discern that the user wants to know the time in London; but the client application itself must then implement the logic to determine the correct time and present it to the user.

In this exercise, you'll use the Azure AI Language service to create a conversational language understand model, and use the Python SDK to implement a client app that uses it.

While this exercise is based on Python, you can develop conversational understanding applications using multiple language-specific SDKs.

Module assessment

1. Your app must interpret a command such as "turn on the light" or "switch the light on". What do these phrases represent in a language model? **Utterances**
2. Your app must interpret a command to book a flight to a specified city, such as "Book a flight to Paris." How should you model the city element of the command? **As an entity**
3. Your language model needs to detect an email when present in an utterance. What is the simplest way to extract that email? **Use prebuilt entity components**

Summary

In this module, you learned how to create a conversational language understanding model.

Now that you've completed this module, you can:

- Provision an Azure AI Language resource
- Define **intents, entities, and utterances**
- Use patterns to differentiate similar utterances
- Use pre-built entity components
- Train, test, publish, and review a model

To learn more about language understanding, see the [Azure AI Language documentation](#).

Create a custom text classification solution

The Azure AI Language service enables processing of natural language to use in your own app. Learn how to build a **custom text classification** project.

Learning objectives

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

- Understand **types of classification projects**
- Build a custom text classification project
- Tag data, train, and deploy a model
- Submit classification tasks from your own app

Introduction

Natural language processing (NLP) is one of the most common AI problems, where software must **interpret text or speech in the natural form that humans use**. Part of NLP is the ability to classify text, and Azure provides ways to **classify text including sentiment, language, and custom categories defined by the user**.

In this module, you'll learn how to use the Azure AI Language service to classify text into custom groups.

Understand types of classification projects

Custom text classification assigns labels, which in the Azure AI Language service is a class that the developer defines, **to text files**. For example, a video game summary might be classified as "Adventure", "Strategy", "Action" or "Sports".

Custom text classification falls into two types of projects:

- **Single label classification** - you can **assign only one class to each file**. Following the above example, a video game summary could only be classified as "Adventure" or "Strategy".
- **Multiple label classification** - you can **assign multiple classes to each file**. This type of project would allow you to classify a video game summary as "Adventure" or "Adventure and Strategy".

When creating your custom text classification project, you can specify which project you want to build.

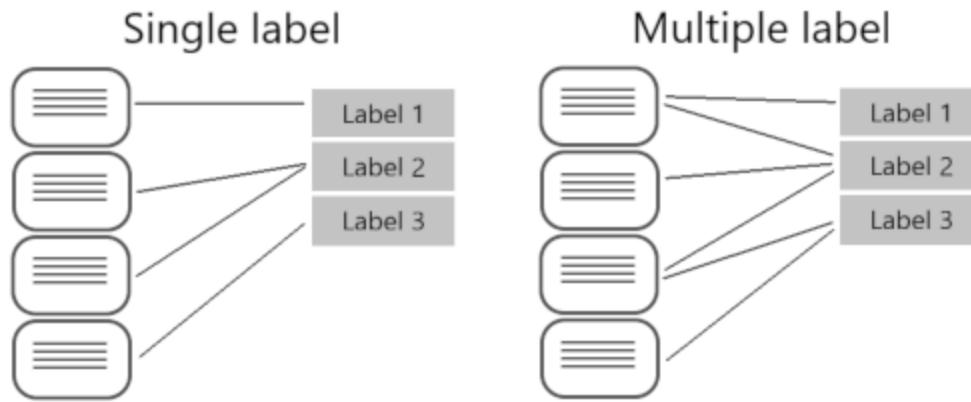
Single vs. multiple label projects

Beyond the ability to put files into multiple classifications, **the key differences with multiple label classification projects are 1) labeling, 2) considerations for improving your model, and 3) the API payload for classification tasks**.

Labeling data

In single label projects, each file is assigned one class during the labeling process; class assignment in Azure AI Language only allows you to select one class.

When labeling multiple label projects, you can assign as many classes that you want per file. The impact of the added complexity means your data has to remain clear and provide a good distribution of possible inputs for your model to learn from.



Labeling data correctly, especially for multiple label projects, is directly correlated with how well your model performs. The higher the quality, clarity, and variation of your data set is, the more accurate your model will be.

Evaluating and improving your model

Measuring predictive performance of your model goes beyond how many predictions were correct. **Correct classifications are when the actual label is x and the model predicts a label x**. In the real

world, documents result in different kinds of errors when a classification isn't correct:

- **False positive** - model predicts x, but the file isn't labeled x.
- **False negative** - model doesn't predict label x, but the file in fact is labeled x.

These metrics are translated into three measures provided by Azure AI Language:

- **Recall** - Of all the actual labels, how many were identified; **the ratio of true positives to all that was labeled**.
- **Precision** - How many of the predicted labels are correct; **the ratio of true positives to all identified positives**.
- **F1 Score** - A function of recall and precision, intended to provide a single score to maximize for a balance of each component

Tip: Learn more about the [Azure AI Language evaluation metrics](#), including exactly how these metrics are calculated.

With a single label project, you can identify which classes aren't classified as well as others and find more quality data to use in training your model. For multiple label projects, figuring out quality data becomes more complex due to the matrix of possible permutations of combined labels.

For example, let's say your model is correctly classifying "Action" games and some "Action and Strategy" games, but failing at "Strategy" games. To improve your model, you'll want to find more high quality and varied summaries for both "Action and Strategy" games, as well as "Strategy" games to teach your model how to differentiate the two. This challenge increases exponentially with more possible classes your model is classifying into.

API payload

Azure AI Language provides a REST API to build and interact with your model, **using a JSON body to specify the request**. This API is abstracted into multiple language-specific SDKs, however for this module we'll focus our examples on the base REST API.

To submit a classification task, the API requires the JSON body to specify which task to execute. You'll learn more about the REST API in the next unit, but worth familiarizing yourself with parts of the required body.

Single label classification models specify a project type of `customSingleLabelClassification`

```
{
  "projectFileVersion": "<API-VERSION>",
  "stringIndexType": "Utf16CodeUnit",
  "metadata": {
    " projectName": "<PROJECT-NAME>",
    "storageInputContainerName": "<CONTAINER-NAME>",
    "projectKind": "customSingleLabelClassification",
    "description": "Trying out custom single label text classification",
    "language": "<LANGUAGE-CODE>",
    "multilingual": true,
    "settings": {}
  },
  "assets": {
    "projectKind": "customSingleLabelClassification",
    "classes": [
      {
        "category": "Class1"
      },
      {
        "category": "Class2"
      }
    ],
    "documents": [
      {
        "location": "<DOCUMENT-NAME>",
        "language": "<LANGUAGE-CODE>",
        "dataset": "<DATASET>",
        "class": {
          "category": "Class2"
        }
      },
      {
        "location": "<DOCUMENT-NAME>",
        "language": "<LANGUAGE-CODE>",
        "dataset": "<DATASET>",
        "class": {
          "category": "Class1"
        }
      }
    ]
  }
}
```

Multiple label classification models specify a project type of `CustomMultiLabelClassification`

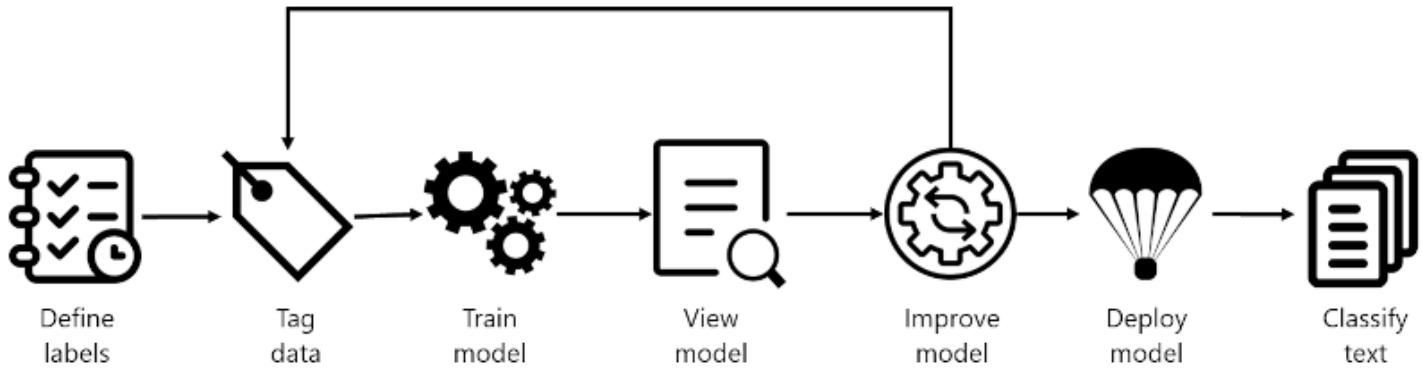
```
{
  "projectFileVersion": "<API-VERSION>",
  "stringIndexType": "Utf16CodeUnit",
  "metadata": {
    " projectName": "<PROJECT-NAME>",
    "storageInputContainerName": "<CONTAINER-NAME>",
    "projectKind": "customMultiLabelClassification",
    "description": "Trying out custom multi label text classification",
    "language": "<LANGUAGE-CODE>",
    "multilingual": true,
    "settings": {}
  },
  "assets": {
    "projectKind": "customMultiLabelClassification",
    "classes": [
      {
        "category": "Class1"
      },
      {
        "category": "Class2"
      }
    ],
    "documents": [
      {
        "location": "<DOCUMENT-NAME>",
        "language": "<LANGUAGE-CODE>",
        "dataset": "<DATASET>",
        "classes": [
          {
            "category": "Class1"
          },
          {
            "category": "Class2"
          }
        ]
      },
      {
        "location": "<DOCUMENT-NAME>",
        "language": "<LANGUAGE-CODE>",
        "dataset": "<DATASET>",
        "classes": [
          {
            "category": "Class2"
          }
        ]
      }
    ]
  }
}
```

```
        }  
    ]  
}  
]  
}  
}  
}
```

Understand how to build text classification projects

Custom text classification projects are your workspace to build, train, improve, and deploy your classification model. **You can work with your project in two ways: through Language Studio and via the REST API.** Language Studio is the GUI that will be used in the lab, but the REST API has the same functionality. Regardless of which method you prefer, the steps for developing your model are the same.

Azure AI Language project life cycle



- **Define labels:** Understanding the data you want to classify, identify the possible labels you want to categorize into. In our video game example, our labels would be "Action", "Adventure", "Strategy", and so on.
- **Tag data:** Tag, or label, your existing data, specifying the label or labels each file falls under. Labeling data is important since it's how your model will learn how to classify future files. Best practice is to have clear differences between labels to avoid ambiguity, and provide good examples of each label for the model to learn from. For example, we'd label the game "Quest for the Mine Brush" as "Adventure", and "Flight Trainer" as "Action".

- **Train model:** Train your model with the labeled data. Training will teach our model what types of video game summaries should be labeled which genre.
- **View model:** After your model is trained, view the results of the model. **Your model is scored between 0 and 1, based on the precision and recall of the data tested.** Take note of which genre didn't perform well.
- **Improve model:** Improve your model by seeing which classifications failed to evaluate to the right label, see your label distribution, and find out what data to add to improve performance. For example, you might find your model mixes up "Adventure" and "Strategy" games. Try to find more examples of each label to add to your dataset for retraining your model.
- **Deploy model:** Once your model performs as desired, deploy your model to make it available via the API. Your model might be named "GameGenres", and once deployed can be used to classify game summaries.
- **Classify text:** Use your model for classifying text. The lab covers how to use the API, and you can view the [API reference](#)

How to split datasets for training

When labeling your data, you can specify which dataset you want each file to be:

- **Training** - The training dataset is used to actually train the model; the data and labels provided are fed into the machine learning algorithm to teach your model what data should be classified to which label. The training dataset will be the larger of the two datasets, recommended to be about **80% of your labeled data**.
- **Testing** - The testing dataset is labeled data used to verify your model after it's trained. Azure will take the data in the testing dataset, submit it to the model, and compare the output to how you labeled your data to determine how well the model performed. The result of that comparison is how your model gets scored and helps you know how to improve your predictive performance.

During the **Train model** step, there are two options for how to train your model.

- **Automatic split** - Azure takes all of your data, splits it into the specified percentages randomly, and applies them in training the model. **This option is best when you have a larger dataset, data is naturally more consistent, or the distribution of your data extensively covers your classes.**
- **Manual split** - Manually specify which files should be in each dataset. When you submit the training job, the Azure AI Language service will tell you the split of the dataset and the distribution. **This split is best used with smaller datasets to ensure the correct distribution of classes and variation in data are present to correctly train your model.**

To use the automatic split, put all files into the training dataset when labeling your data (this option is the default). To use the manual split, specify which files should be in testing versus training during the labeling of your data.

Deployment options

Azure AI Language allows each project to create both **multiple models** and **multiple deployments**, each with their own unique name. Benefits include ability to:

- Test two models side by side
- Compare how the split of datasets impact performance
- Deploy multiple versions of your model

Note: Each project has a limit of ten deployment names

During deployment you can choose the name for the deployed model, which can then be selected when submitting a classification task:

```
<...>
  "tasks": [
    {
      "kind": "CustomSingleLabelClassification",
      "taskName": "MyTaskName",
      "parameters": {
        "projectName": "MyProject",
        "deploymentName": "MyDeployment"
      }
    }
  ]
<...>
```

Using the REST API

The REST API available for the Azure AI Language service allows for CLI development of Azure AI Language projects in the same way that Language Studio provides a user interface for building projects. Language Studio is explored further in this module's lab.

Pattern of using the API

The API for the Azure AI Language service operates **asynchronously** for most calls. In each step we **submit a request to the service first**, then **check back with the service via a subsequent call to**

get the status or result.

With each request, a header is required to authenticate your request:

Key	Value
Ocp-Apim-Subscription-Key	The key to your Azure AI Language resource

Submit initial request

The URL to submit the request to varies on which step you are on, but all are prefixed with the `endpoint` provided by your Azure AI Language resource.

For example, to train a model, you would create a POST to the URL that would look something like the following:

`<YOUR-ENDPOINT>/language/analyze-text/projects/<PROJECT-NAME>/:train?api-version=<API-VERSION>`

Placeholder	Value	Example
YOUR-ENDPOINT	The endpoint for your API request	<code>https://<your-custom-resource>.cognitiveservices.azure.com</code>
PROJECT-NAME	The name for your project (value is case-sensitive)	<code>myProject</code>

The following body would be attached to the request:

```
{
  "modelLabel": "<MODEL-NAME>",
  "trainingConfigVersion": "<CONFIG-VERSION>",
  "evaluationOptions": {
    "kind": "percentage",
    "trainingSplitPercentage": 80,
    "testingSplitPercentage": 20
  }
}
```

Key	Value
YOUR-MODEL	Your model name.
trainingConfigVersion	The model version to use to train your model.
runValidation	Boolean value to run validation on the test set.
evaluationOptions	Specifies evaluation options.
kind	Specifies data split type. Can be <code>percentage</code> if you're using an automatic split, or <code>set</code> if you manually split your dataset
testingSplitPercentage	Required integer field only if <code>type</code> is <code>percentage</code> . Specifies testing split.
trainingSplitPercentage	Required integer field only if <code>type</code> is <code>percentage</code> . Specifies training split.

The response to the above request will be a `202`, meaning the request was successful. Grab the location value from the response headers, which will look similar to the following URL:

`<ENDPOINT>/language/analyze-text/projects/<PROJECT-NAME>/train/jobs/<JOB-ID>?api-version=<API-VI`

Key	Value
JOB-ID	Identifier for your request

This URL is used in the next step to get the training status.

Get training status

To get the training status, use the URL from the header of the request response to submit a `GET` request, with same header that provides our Azure AI Language service key for authentication. The response body will be similar to the following JSON:

```
{  
  "result": {  
    "modelLabel": "<MODEL-NAME>",  
    "trainingConfigVersion": "<CONFIG-VERSION>",  
    "estimatedEndDateTime": "2023-05-18T15:47:58.8190649Z",  
    "trainingStatus": {  
      "percentComplete": 3,  
      "startDateTime": "2023-05-18T15:45:06.8190649Z",  
      "status": "running"  
    },  
    "evaluationStatus": {  
      "percentComplete": 0,  
      "status": "notStarted"  
    }  
  },  
  "jobId": "<JOB-ID>",  
  "createdDateTime": "2023-05-18T15:44:44Z",  
  "lastUpdatedDateTime": "2023-05-18T15:45:48Z",  
  "expirationDateTime": "2023-05-25T15:44:44Z",  
  "status": "running"  
}
```

Training a model can take some time, so periodically check back at this status URL until the response `status` returns `succeeded`. Once the training has succeeded, you can **view, verify, and deploy your model**.

Consuming a deployed model

Using the model to classify text follows the same pattern as outlined above, with a `POST` request submitting the job and a `GET` request to retrieve the results.

Submit text for classification

To use your model, submit a `POST` to the analyze endpoint at the following URL:

`<ENDPOINT>/language/analyze-text/jobs?api-version=<API-VERSION>`

Placeholder	Value	Example
YOUR-ENDPOINT	The endpoint for your API request	<code>https://<your-custom-resource>.cognitiveservices.azure.com</code>

Important: Remember to include your resource key in the header for `Ocp-Apim-Subscription-Key`

The following JSON structure would be attached to the request:

```
{
  "displayName": "Classifying documents",
  "analysisInput": {
    "documents": [
      {
        "id": "1",
        "language": "<LANGUAGE-CODE>",
        "text": "Text1"
      },
      {
        "id": "2",
        "language": "<LANGUAGE-CODE>",
        "text": "Text2"
      }
    ],
    "tasks": [
      {
        "kind": "<TASK-REQUIRED>",
        "taskName": "<TASK-NAME>",
        "parameters": {
          "projectName": "<PROJECT-NAME>",
          "deploymentName": "<DEPLOYMENT-NAME>"
        }
      }
    ]
  }
}
```

Key	Value
TASK-REQUIRED	Which task you're requesting. The task is <code>CustomMultiLabelClassification</code> for multiple label projects, or <code>CustomSingleLabelClassification</code> for single

Key	Value
	label projects
LANGUAGE-CODE	The language code such as en-us .
TASK-NAME	Your task name.
PROJECT-NAME	Your project name.
DEPLOYMENT-NAME	Your deployment name.

The response to the above request will be a 202 , meaning the request was successful. Look for the operation-location value in the response headers, which will look something like the following URL:

<ENDPOINT>/language/analyze-text/jobs/<JOB-ID>?api-version=<API-VERSION>

Key	Value
YOUR-ENDPOINT	The endpoint for your API request
JOB-ID	Identifier for your request

This URL is used to get your task results.

Get classification results

Submit a GET request to the endpoint from the previous request, with the same header for authentication. The response body will be similar to the following JSON:

```
{
  "createdDateTime": "2023-05-19T14:32:25.578Z",
  "displayName": "MyJobName",
  "expirationDateTime": "2023-05-19T14:32:25.578Z",
  "jobId": "xxxx-xxxxxx-xxxxx-xxxx",
  "lastUpdateDateTime": "2023-05-19T14:32:25.578Z",
  "status": "succeeded",
  "tasks": {
    "completed": 1,
    "failed": 0,
    "inProgress": 0,
    "total": 1,
    "items": [
      {
        "kind": "customSingleClassificationTasks",
        "taskName": "Classify documents",
        "lastUpdateDateTime": "2022-10-01T15:01:03Z",
        "status": "succeeded",
        "results": {
          "documents": [
            {
              "id": "<DOC-ID>",
              "class": [
                {
                  "category": "Class_1",
                  "confidenceScore": 0.0551877357
                }
              ],
              "warnings": []
            }
          ],
          "errors": [],
          "modelVersion": "2022-04-01"
        }
      }
    ]
  }
}
```

The classification result is within the items array's `results` object, for each document submitted.

Exercise - Classify text

In this exercise, you use Azure AI Language to build a custom text classification model.

Custom text classification

Azure AI Language provides several NLP capabilities, including the key phrase identification, text summarization, and sentiment analysis. The Language service also provides custom features like **custom question answering and custom text classification**.

To test the custom text classification of the Azure AI Language service, you'll configure the model using Language Studio then use a Python application to test it.

While this exercise is based on Python, you can develop text classification applications using multiple language-specific SDKs; including:

- [Azure AI Text Analytics client library for Python](#)
- [Azure AI Text Analytics client library for .NET](#)
- [Azure AI Text Analytics client library for JavaScript](#)

Module assessment

1. You want to train a model to classify book summaries by their genre, and some of your favorite books are both *mystery* and *thriller*. Which type of project should you build? **A multiple label classification project.**
2. You just got notification your training job is complete. What is your next step? **View your model details.**
3. You want to submit a classification task via the API. How do you get the results of the classification? **Call the URL provided in the header of the request response.**

Summary

In this module, you learned about custom text classification and how to build a text classification service.

Now that you've completed this module, you can:

- Understand types of classification projects.
- Build a custom text classification project.
- Tag data, train, and deploy a model.
- Submit classification tasks from your own app.

To learn more about the Azure AI Language service, see the [Azure AI Language service documentation](#).

Custom named entity recognition

Build a custom entity recognition solution to **extract entities from unstructured documents**.

Learning objectives

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

- Understand tagging entities in extraction projects
- Understand how to **build entity recognition projects**

Introduction

Custom **named entity recognition (NER)**, otherwise known as **custom entity extraction**, is one of the many features for natural language processing (NLP) offered by Azure AI Language service.

Custom NER enables developers to **extract predefined entities from text documents**, without those documents being in a known format - such as legal agreements or online ads.

An entity is a **person, place, thing, event, skill, or value**.

In this module, you'll learn how to use the **Azure AI Language service** to extract entities from unstructured documents.

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

- Understand custom named entities and how they're labeled.
- Build **a custom named entity extraction project**.
- Label data, train, and deploy an **entity extraction model**.
- Submit extraction tasks from your own app.

Understand custom named entity recognition

Custom NER is an Azure API service that looks at documents, *identifies, and extracts user defined entities*. These entities could be anything from *names and addresses from bank statements to knowledge mining to improve search results*.

Custom NER is part of Azure AI Language in Azure AI services.

Custom vs built-in NER

Azure AI Language provides certain built-in entity recognition, to recognize things such as a person, location, organization, or URL. Built-in NER allows you to set up the service with minimal configuration, and extract entities. **To call a built-in NER, create your service and call the endpoint for that NER service** like this:

```
<YOUR-ENDPOINT>/language/analyze-text/jobs?api-version=<API-VERSION>
```

Placeholder	Value	Example
YOUR-ENDPOINT	The endpoint for your API request	https://.cognitiveservices.azure.com
API-VERSION	The version of the API you are calling	2023-05-01

The body of that call will contain the document(s) the entities are extracted from, and **the headers** contain your service **key**.

The response from the call above contains an array of entities recognized, such as:

```
<...>
"entities": [
  {
    "text": "Seattle",
    "category": "Location",
    "subcategory": "GPE",
    "offset": 45,
    "length": 7,
    "confidenceScore": 0.99
  },
  {
    "text": "next week",
    "category": "DateTime",
    "subcategory": "DateRange",
    "offset": 104,
    "length": 9,
    "confidenceScore": 0.8
  }
]
<...>
```

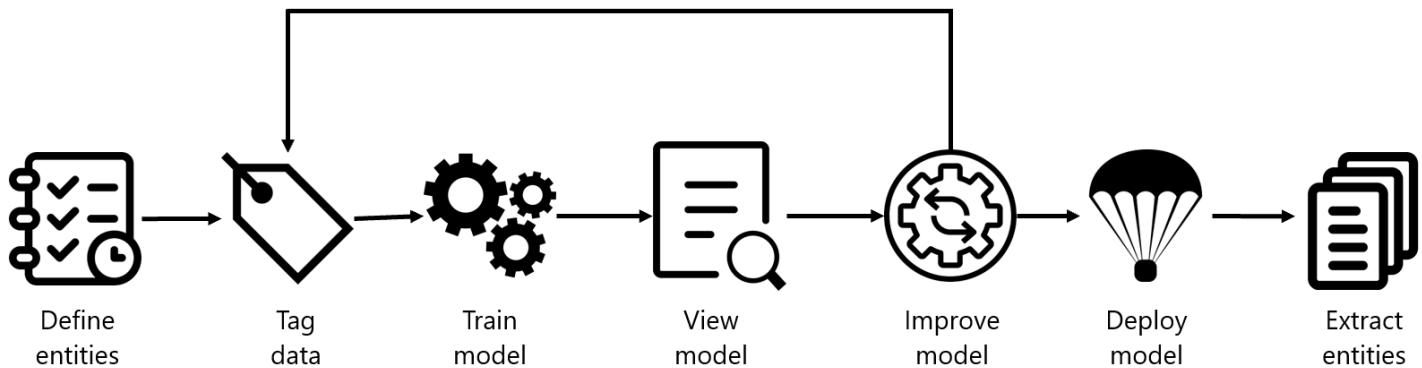
Examples of when to use the **built-in NER** include finding **locations, names, or URLs in long text documents**.

Tip: A full list of recognized entity categories is available in the [NER docs](#).

Custom NER, which is the focus of the rest of this module, is available when the entities you want to extract aren't part of the built-in service or you only want to extract specific entities. You can make your custom NER model as simple or complex as is required for your app.

Examples of when you'd want custom NER include *specific legal or bank data, knowledge mining to enhance catalog search, or looking for specific text for audit policies*. Each one of these projects requires a specific set of entities and data it needs to extract.

Azure AI Language project life cycle



Creating an entity extraction model typically follows a similar path to most Azure AI Language service features:

- **Define entities:** Understanding the data and entities you want to identify, and try to make them as clear as possible. For example, defining exactly which parts of a bank statement you want to extract.
- **Tag data:** *Label, or tag, your existing data, specifying what text in your dataset corresponds to which entity.* This step is important to do accurately and completely, as any wrong or missed labels will reduce the effectiveness of the trained model. A good variation of possible input documents is useful. For example, label bank name, customer name, customer address, specific loan or account terms, loan or account amount, and account number.
- **Train model:** Train your model once your entities are labeled. Training teaches your model how to recognize the entities you label.
- **View model:** *After your model is trained, view the results of the model.* This page includes a **score of 0 to 1 that is based on the precision and recall of the data tested.** You can see which entities worked well (such as customer name) and which entities need improvement (such as account number).
- **Improve model:** Improve your model by seeing which entities failed to be identified, and which entities were incorrectly extracted. Find out what data needs to be added to your model's training to improve performance. This page shows you how entities failed, and which entities (such as account number) need to be differentiated from other similar entities (such as loan amount).
- **Deploy model:** Once your model performs as desired, deploy your model to make it available via the API. In our example, you can send requests to the model when it's deployed to extract bank statement entities.
- **Extract entities:** Use your model for extracting entities. The lab covers how to use the API, and you can view the [API reference](#) for more details.

Considerations for data selection and refining entities

For the best performance, you'll need to use both **high quality data** to train the model and **clearly defined entity types**.

High quality data will let you spend less time refining and yield better results from your model.

- **Diversity** - use as diverse of a dataset as possible without losing the real-life distribution expected in the real data. You'll want to **use sample data from as many sources as possible**, each with their own formats and number of entities. It's best to have your dataset represent as many different sources as possible.
- **Distribution** - use the appropriate distribution of document types. A more diverse dataset to train your model will help your model avoid learning incorrect relationships in the data.
- **Accuracy** - **use data that is as close to real world data as possible**. Fake data works to start the training process, but it likely will differ from real data in ways that can cause your model to not extract correctly.

Entities need to also be carefully considered, and defined as distinctly as possible. Avoid ambiguous entities (such as two names next to each other on a bank statement), as it will make the model struggle to differentiate. If having some ambiguous entities is required, make sure to have more examples for your model to learn from so it can understand the difference.

Keeping your entities distinct will also go a long way in helping your model's performance. For example, trying to extract something like "Contact info" that could be a phone number, social media handle, or email address would require several examples to correctly teach your model. Instead, **try to break them down into more specific entities** such as "Phone", "Email", and "Social media" and let the model label whichever type of contact information it finds.

How to extract entities

To submit an extraction task, the API requires the JSON body to specify which task to execute. For custom NER, the task for the JSON payload is `CustomEntityRecognition`.

Your payload will look similar to the following JSON:

```
{
  "displayName": "string",
  "analysisInput": {
    "documents": [
      {
        "id": "doc1",
        "text": "string"
      },
      {
        "id": "doc2",
        "text": "string"
      }
    ]
  },
  "tasks": [
    {
      "kind": "CustomEntityRecognition",
      "taskName": "MyRecognitionTaskName",
      "parameters": {
        "projectName": "MyProject",
        "deploymentName": "MyDeployment"
      }
    }
  ]
}
```

Project limits

The Azure AI Language service enforces the following restrictions:

- **Training** - at least 10 files, and not more than 100,000
- **Deployments** - 10 deployment names per project
- **APIs**
 - **Authoring** - this API creates a project, trains, and deploys your model. Limited to 10 POST and 100 GET per minute
 - **Analyze** - this API does the work of actually extracting the entities; it requests a task and retrieves the results. Limited to 20 GET or POST
- **Projects** - only 1 storage account per project, 500 projects per resource, and 50 trained models per project
- **Entities** - each entity can be up to 500 characters. You can have up to 200 entity types.

See the [Service limits for Azure AI Language](#) page for detailed information.

Label your data

Labeling, or tagging, your data correctly is an important part of the process to create a custom entity extraction model. Labels identify examples of specific entities in text used to train the model. Three things to focus on are:

- **Consistency** - *Label your data the same way across all files for training.* Consistency allows your model to learn without any conflicting inputs.
- **Precision** - *Label your entities consistently, without unnecessary extra words.* Precision ensures only the correct data is included in your extracted entity.
- **Completeness** - *Label your data completely, and don't miss any entities.* Completeness helps your model always recognize the entities present.

Data labeling ✓ Saved

Select a document to annotate its text with entity labels. After labeling the documents and adding them to training or testing sets, you'll be ready to create a model with this data in [Training](#).

Single document view ▾ Document name: Ad 1.txt

The screenshot shows a tooltip for the entity 'ItemForSale'. The tooltip contains the following text:
1 face cord of firewood for sale
ItemForSale
Dry, seasoned hardwood
\$90
Pri...
Search for an entity
ItemForSale
Price
Location
Remove label

On the right side of the interface, there is a sidebar titled 'Labels' which lists three categories: 'ItemForSale ...', 'Price (12)', and 'Location (12)'. Each category has a delete icon next to it.

How to label your data

Language Studio is the most straight forward method for labeling your data. Language Studio allows you to *see the file, select the beginning and end of your entity, and specify which entity it is.*

Each label that you identify gets saved into a file that lives in your storage account with your dataset, in an auto-generated JSON file. This file then gets used by the model to learn how to extract custom entities. It's possible to provide this file when creating your project (if you're importing the same labels from a different project, for example) however it must be in the [Accepted custom NER data formats](#). For example:

```
{  
  "projectFileVersion": "{DATE}",  
  "stringIndexType": "Utf16CodeUnit",  
  "metadata": {  
    "projectKind": "CustomEntityRecognition",  
    "storageInputContainerName": "{CONTAINER-NAME}",  
    " projectName": "{PROJECT-NAME}",  
    "multilingual": false,  
    "description": "Project-description",  
    "language": "en-us",  
    "settings": {}  
  },  
  "assets": {  
    "projectKind": "CustomEntityRecognition",  
    "entities": [  
      {  
        "category": "Entity1"  
      },  
      {  
        "category": "Entity2"  
      }  
    ],  
    "documents": [  
      {  
        "location": "{DOCUMENT-NAME}",  
        "language": "{LANGUAGE-CODE}",  
        "dataset": "{DATASET}",  
        "entities": [  
          {  
            "regionOffset": 0,  
            "regionLength": 500,  
            "labels": [  
              {  
                "category": "Entity1",  
                "offset": 25,  
                "length": 10  
              },  
              {  
                "category": "Entity2",  
                "offset": 120,  
                "length": 8  
              }  
            ]  
          ]  
        ]  
      ]  
    ]  
  }  
}
```

```
        }
    ],
},
{
  "location": "{DOCUMENT-NAME}",
  "language": "{LANGUAGE-CODE}",
  "dataset": "{DATASET}",
  "entities": [
    {
      "regionOffset": 0,
      "regionLength": 100,
      "labels": [
        {
          "category": "Entity2",
          "offset": 20,
          "length": 5
        }
      ]
    }
  ]
}
}
```

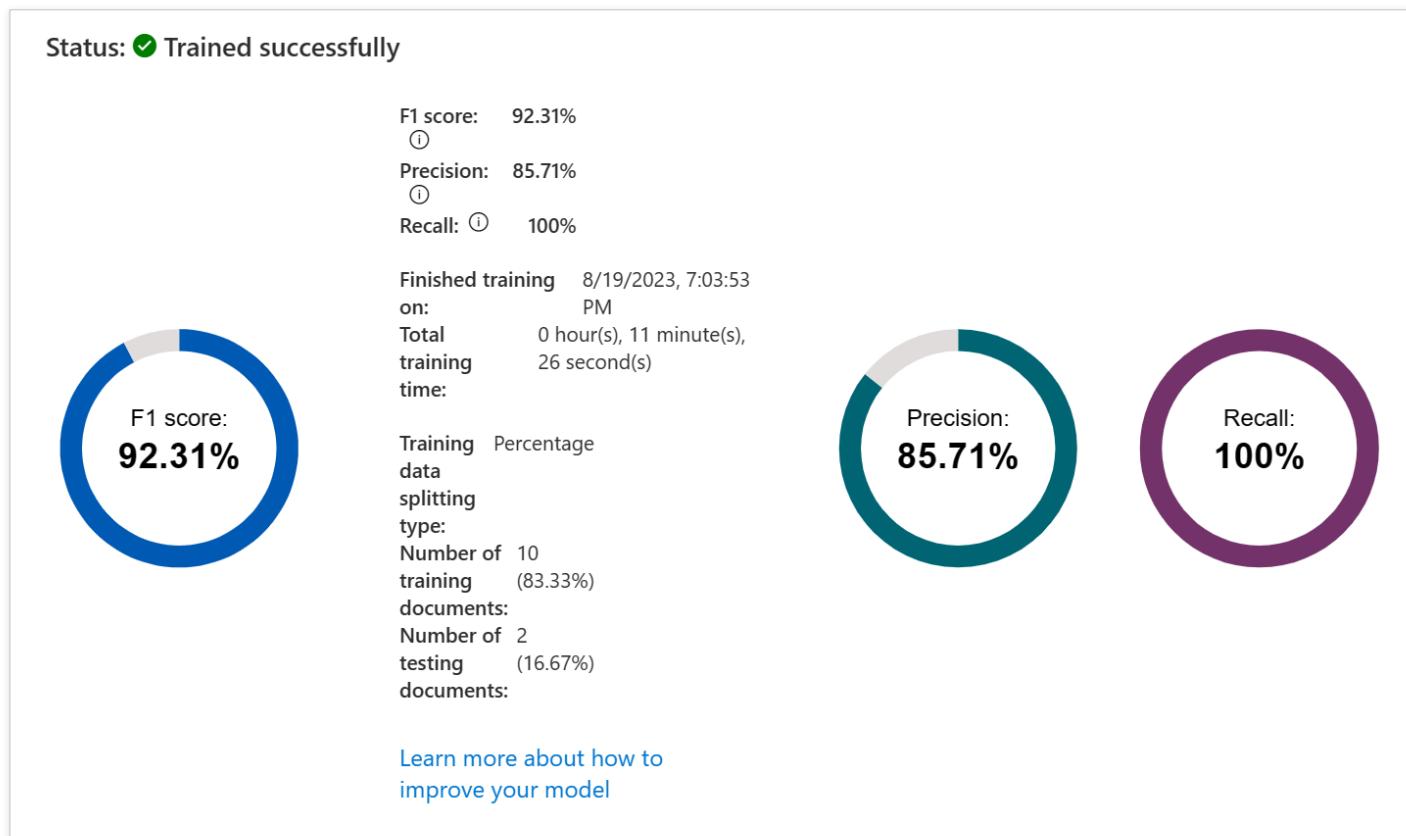
Field	Description
documents	Array of labeled documents
location	Path to file within container connected to the project
language	Language of the file
entities	Array of present entities in the current document
regionOffset	Inclusive character position for start of text
regionLength	Length in characters of the data used in training
category	Name of entity to extract
labels	Array of labeled entities in the files
offset	Inclusive character position for start of entity

Field	Description
length	Length in characters of the entity
dataset	Which dataset the file is assigned to

Train and evaluate your model

Training and evaluating your model is *an iterative process of adding data and labels to your training dataset to teach the model more accurately*. To know what types of data and labels need to be improved, Language Studio provides **scoring** in the View model details page on the left hand pane.

[Overview](#) [Entity type performance](#) [Test set details](#) [Dataset distribution](#) [Confusion matrix](#)



Individual entities and your overall model score are broken down into three metrics to explain how they're performing and where they need to improve.

Metric	Description
Precision	The ratio of successful entity recognitions to all attempted recognitions. A high score means that as long as the entity is recognized, it's labeled correctly. Precision = $TP / (TP + FP)$
Recall	The ratio of successful entity recognitions to the actual number of entities in the document. A high score means it finds the entity or entities well, regardless of if it assigns them the right label. Recall = $TP / (TP + FN)$
F1 score	Combination of precision and recall providing a single scoring metric. F1 score = $2 \times P \times R / (P + R)$

Scores are available both per entity and for the model as a whole. You may find an entity scores well, but the whole model doesn't.

How to interpret metrics

Ideally we want our model to score well in both precision and recall, which means the entity recognition works well.

If both metrics have a low score, it means the model is both struggling to recognize entities in the document, and when it does extract that entity, it doesn't assign it the correct label with high confidence.

If precision is low but recall is high, it means that the model recognizes the entity well but doesn't label it as the correct entity type.

If precision is high but recall is low, it means that the model doesn't always recognize the entity, but when the model extracts the entity, the correct label is applied.

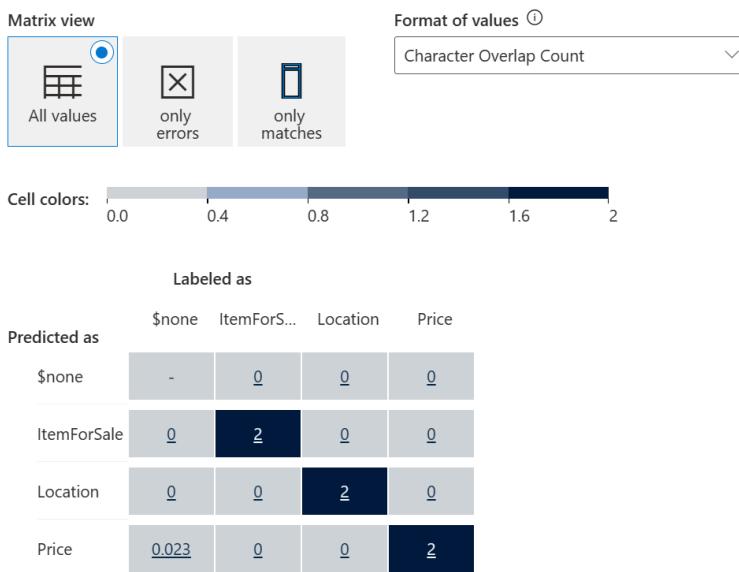
Confusion matrix

On the same **View model details** page, there's another tab on the top for the **Confusion matrix**. This view provides a visual table of all the entities and how each performed, giving a complete view of the model and where it's falling short.

A confusion matrix is an N x N matrix used for evaluating the performance of a extraction model, where N is the number of target entities. The matrix compares the actual target values with those predicted by the machine learning model to show how well the extraction model is performing and what kinds of errors it is making.

All correct predictions are located in the diagonal of the table and errors are values outside of the diagonal. Other numbers in the row show where it was incorrectly predicted as other entities. To see how to improve your model, check the recommendations in the [Overview tab](#).

[Learn how to read a confusion matrix.](#)



The confusion matrix allows you to visually identify where to add data to improve your model's performance.

Exercise - Extract custom entities

In this exercise, you use Azure AI Language to build a custom named entity recognition model.

Extract custom entities

In addition to other natural language processing capabilities, Azure AI Language Service enables you to define custom entities, and extract instances of them from text.

To test the custom entity extraction, we'll create a model and train it through Azure AI Language Studio, then use a command line application to test it.

Module assessment

1. You've trained your model and you're seeing that it doesn't recognize your entities. What metric score is likely low to indicate that issue? **Recall**
2. You just finished labeling your data. How and where is that file stored to train your model? **JSON file, in my storage account container for the project.**
3. You train your model with only one source of documents, even though real extraction tasks will come from several sources. What data quality metric do you need to increase? **Diversity**

Summary

In this module, you learned about **custom named entity recognition (NER)** and how to extract entities.

In this module, you learned how to:

- Understand custom named entities and how they're labeled.
- Build a Language service project.
- Label data, train, and deploy an entity extraction model.
- Submit extraction tasks from your own app.

To learn more about Azure AI Language, see the [Azure AI Language documentation](#).

Translate text with Azure AI Translator service

The Translator service enables you to create intelligent apps and services that can **translate text between languages**.

Learning objectives

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

- Provision a **Translator** resource
- Understand **language detection, translation, and transliteration**
- Specify translation options
- Define custom translations

Introduction

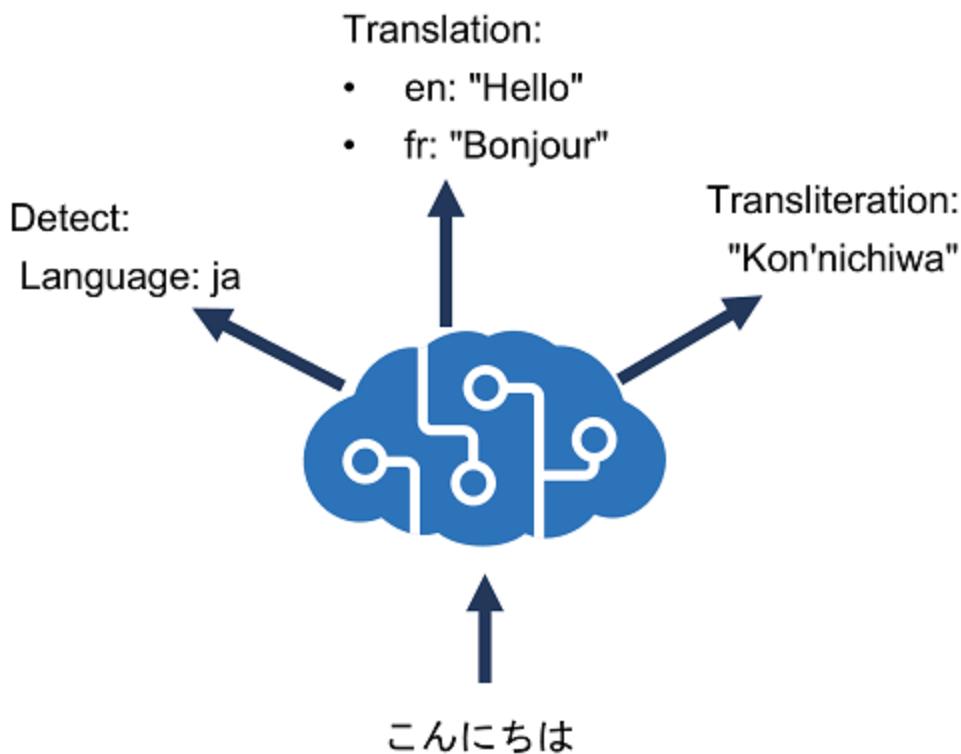
There are many commonly used languages throughout the world, and the ability to exchange information between speakers of different languages is often a critical requirement for global solutions.

The Azure AI Translator provides an API for **translating text between 90 supported languages**.

Provision an Azure AI Translator resource

Azure AI Translator provides a multilingual text translation API that you can use for:

- *Language detection.*
- *One-to-many translation.*
- *Script transliteration* (converting text from its native script to an alternative script).



Azure resource for Azure AI Translator

To use the Azure AI Translator service, you must provision a resource for it in your Azure subscription. You can provision a **single-service Azure AI Translator resource**, or you can use the **Text Translation API in a multi-service Azure AI Services resource**.

After you provision a suitable resource in your Azure subscription, you can use the location where you deployed the resource and one of its subscription keys to call the Azure AI Translator APIs from your code. You can call the APIs by **submitting requests in JSON format to the REST interface**, or by using any of the available **programming language-specific SDKs**.

Note: The code examples in the subsequent units in this module show the `JSON` requests and responses exchanged with the `REST` interface. When using an SDK, the `JSON` requests are abstracted by appropriate objects and methods that encapsulate the same data values. You'll get a chance to try the SDK for C# or Python for yourself in the exercise later in the module.

Understand language detection, translation, and

transliteration

Let's explore the capabilities of Azure AI Translator. These capabilities include:

Language detection

You can use the **Detect** function of the **REST API** to detect the language in which text is written.

For example, you could submit the following text to the [endpoint](#) using curl.

Here's the text we want to translate:

```
{ 'Text' : 'こんにちは' }
```

Here's a call using curl to the endpoint to detect the language of our text:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/detect?api-version=3.0" -H "Ocp-Apim-Subscription-Key: your-subscription-key"
```

The response to this request looks as follows, indicating that the text is written in Japanese:

```
[  
 {  
   "language": "ja",  
   "score": 1.0,  
   "isTranslationSupported": true,  
   "isTransliterationSupported": true  
 }  
]
```

Translation

To translate text from one language to another, use the **Translate** function; specifying a *single from parameter to indicate the source language*, and *one or more to parameters to specify the languages into which you want the text translated*.

For example, you could submit the same JSON we previously used to detect the language, specifying a **from parameter** of ja (Japanese) and **two to parameters** with the values en (English) and fr (French). To do this, you'd call:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=ja&to=en&text=Hello%0ABonjour"
```

This would produce the following result:

```
[  
  {"translations": [  
    [{"text": "Hello", "to": "en"},  
     {"text": "Bonjour", "to": "fr"}]  
  ]}  
]
```

Transliteration

Our Japanese text is written using Hiragana script, so rather than translate it to a different language, you may want to transliterate it to a different script - for example to render the text in Latin script (as used by English language text).

To accomplish this, we can submit the Japanese text to the **Transliterate** function with a **fromScript** parameter of Jpan and a **toScript** parameter of Latn:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/transliterate?api-version=3.0&fromScript=Jpan&toScript=Latn&text=Kon'nichiwa"
```

The response would give you the following result:

```
[  
  {  
    "script": "Latn",  
    "text": "Kon'nichiwa"  
  }  
]
```

Specify translation options

The **Translate** function of the API supports numerous parameters that affect the output.

Word alignment

In written English (using Latin script), spaces are used to separate words. However, in some other languages (and more specifically, scripts) this is not always the case.

For example, translating "Smart Services" from en (English) to zh (Simplified Chinese) produces the result "智能服务", and it's difficult to understand the relationship between the characters in the source text and the corresponding characters in the translation. To resolve this problem, you can specify the **includeAlignment** parameter with a value of **true** in your call to produce the following result:

```
[  
  {  
    "translations": [  
      {  
        "text": "智能服务",  
        "to": "zh-Hans",  
        "alignment": {  
          "proj": "0:4-0:1 6:13-2:3"  
        }  
      }  
    ]  
  }  
]
```

These results tell us that characters 0 to 4 in the source correspond to characters 0 to 1 in the translation, while characters 6 to 13 in the source correspond to characters 2 to 3 in the translation.

Sentence length

Sometimes it might be useful to know the **length of a translation**, for example to determine how best to display it in a user interface. You can get this information by setting the **includeSentenceLength** parameter to true.

For example, specifying this parameter when translating the English (en) text "Hello world" to French (fr) produces the following results:

```
[
  {
    "translations": [
      {
        "text": "Salut tout le monde",
        "to": "fr",
        "sentLen": {"srcSentLen": [12], "transSentLen": [20]}
      }
    ]
  }
]
```

Profanity filtering

Sometimes text contains profanities, which you might want to obscure or omit altogether in a translation. You can handle profanities by specifying the **profanityAction** parameter, which can have one of the following values:

- **NoAction**: Profanities are translated along with the rest of the text.
- **Deleted**: Profanities are omitted in the translation.
- **Marked**: Profanities are indicated using the technique indicated in the **profanityMarker** parameter (if supplied). The default value for this parameter is Asterisk, which replaces characters in profanities with "*". As an alternative, you can specify a profanityMarker value of Tag, which causes profanities to be enclosed in XML tags.

For example, translating the English (en) text "JSON is ████ great!" (where the blocked out word is a profanity) to German (de) with a profanityAction of Marked and a profanityMarker of Asterisk produces the following result:

```
[
  {
    "translations": [
      {
        "text": "JSON ist *** erstaunlich.",
        "to": "de"
      }
    ]
  }
]
```

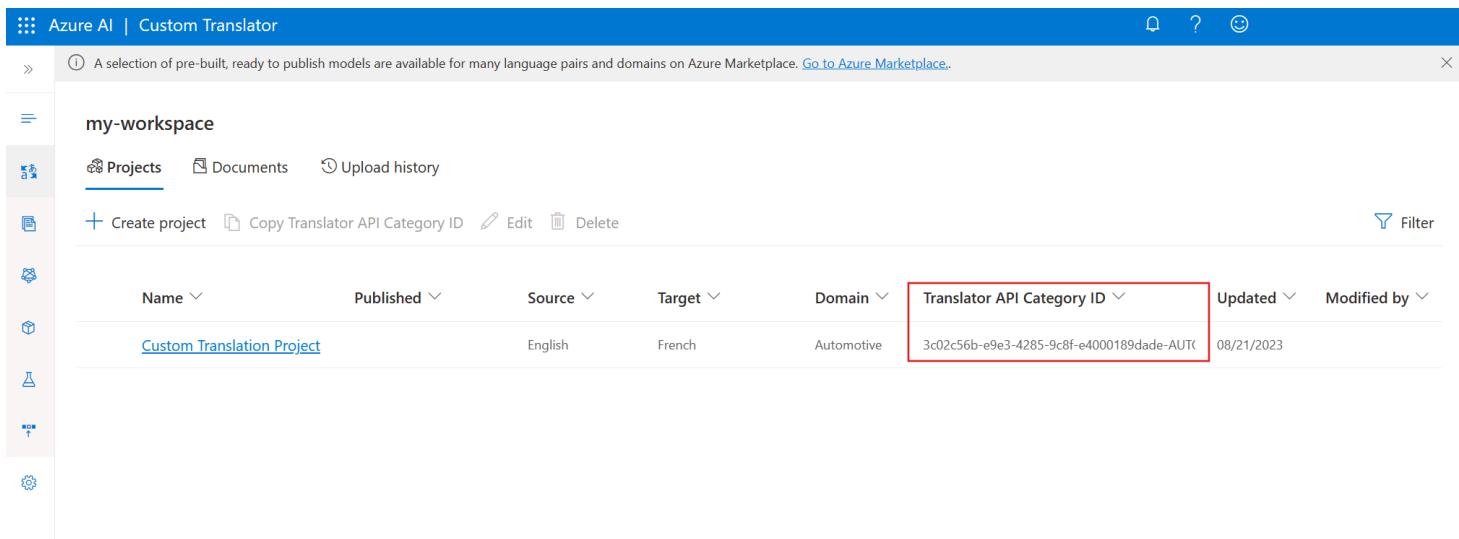
Note: To learn more about the translation options, including some not described here, see the [Azure AI Translator API](#) documentation.

Define custom translations

While the default translation model used by Azure AI Translator is effective for general translation, you may need to develop a **translation solution** for businesses or industries that have **specific vocabularies of terms that require custom translation**.

To solve this problem, you can **create a custom model that maps your own sets of source and target terms for translation**. To create a custom model, use the Custom Translator portal to:

- [Create a workspace](#) linked to your Azure AI Translator resource.
- [Create a project](#).
- [Upload training data files](#) and [train a model](#).
- [Test your model](#) and [publish your model](#).
- Make translation calls to the API.



The screenshot shows the Azure AI | Custom Translator portal interface. At the top, there's a blue header bar with the title 'Azure AI | Custom Translator'. Below the header, a navigation sidebar on the left includes icons for Home, Projects, Documents, Domains, Translation, and Settings. The main content area is titled 'my-workspace'. It displays a table of projects with one entry: 'Custom Translation Project'. The columns in the table are: Name, Published, Source, Target, Domain, Translator API Category ID, Updated, and Modified by. The 'Translator API Category ID' column for the project is highlighted with a red border. The table also shows the values: English as Source, French as Target, Automotive as Domain, and the category ID '3c02c56b-e9e3-4285-9c8f-e4000189dade-AUTC' as the Translator API Category ID. The 'Updated' and 'Modified by' columns show the date '08/21/2023'.

Your custom model is assigned **a unique category Id** (highlighted in the screenshot), which you can specify in translate calls to your **Azure AI Translator resource** by using the category parameter, causing translation to be performed by your custom model instead of the default model.

How to call the API

To initiate a translation, you send a POST request to the following request URL:

<https://api.cognitive.microsofttranslator.com/translate?api-version=3.0>

Your request needs to include a couple of **parameters**:

- `api-version` : The required version of the API.
- `to` : The **target language to translate to**. For example: `to=fr` for French.
- `category` : Your category Id.

Your request must also include a number of required headers:

- `Ocp-Apim-Subscription-Key` . Header for your client key. For example:
`Ocp-Apim-Subscription-Key=<your-client-key>` .
- `Content-Type` . The content type of the payload. The required format is:
`Content-Type: application/json; charset=UTF-8` .

The request body should contain an array that includes a JSON object with a `Text` property that specifies the text that you want to translate:

```
[  
    {"Text": "Where can I find my employee details?"}  
]
```

There are different ways you can send your request to the API, including using the C#, Python, and curl. For instance, to make a quick call, you can send a `POST` request using curl:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=nl"  
-H "Ocp-Apim-Subscription-Key: <your-client-key>"  
-H "Content-Type: application/json; charset=UTF-8"
```

The request above makes a call to translate a sentence from English to Dutch.

Response returned

The response returns a response code of `200` if the request was successful. It also returns a response body that contains the translated text, like this:

```
[  
  {  
    "translations": [  
      {"text": "Waar vind ik mijn personeelsgegevens?", "to": "nl"}  
    ]  
  }  
]
```

If the request wasn't successful, then a number of different status codes may be returned depending on the error type, such as `400` (*missing or invalid query parameters*). See [Response status codes](#) for a full list of codes and their explanation.

Note: For more information about custom translation, see [Quickstart: Build, publish, and translate with custom models](#).

Exercise - Translate text with the Azure AI Translator service

In this exercise, you build an app that translates text between languages.

Translate Text

Azure AI Translator is a service that enables you to translate text between languages. In this exercise, you'll use it to create a simple app that translates input in any supported language to the target language of your choice.

Module assessment

1. What function of Azure AI Translator should you use to convert the Chinese word "你好" to the English word "Hello"? **Translate**
2. What function of Azure AI Translator should you use to convert the Russian word "спасибо" in Cyrillic characters to "spasibo" in Latin characters? **Transliterate**

Summary

In this module, you learned how to:

- Provision an Azure AI Translator resource
- Understand language detection, translation, and transliteration
- Specify translation options
- Define custom translations

To learn more about Azure AI Translator, see the [Azure AI Translator documentation](#).

Create speech-enabled apps with Azure AI services

The **Azure AI Speech service** enables you to build **speech-enabled applications**. This module focuses on using the **speech-to-text and text to speech APIs**, which enable you to create apps that are capable of **speech recognition** and **speech synthesis**.

Learning objectives

In this module, you'll learn how to:

- Provision an Azure resource for the Azure AI Speech service
- Implement *speech recognition* with the *Azure AI Speech to text API*
- Use the *Text to speech API* to implement *speech synthesis*
- Configure audio format and voices
- Use **Speech Synthesis Markup Language (SSML)**

API	Function
Speech to text	Speech recognition
Text to speech	Speech synthesis

Introduction

Azure AI Speech provides APIs that you can use to build **speech-enabled applications**. This includes:

- **Speech to text:** An API that enables **speech recognition** in which your application can accept **spoken input**.
- **Text to speech:** An API that enables **speech synthesis** in which your application can provide **spoken output**.
- **Speech Translation:** An API that you can use to **translate spoken input into multiple languages**.

- **Keyword Recognition:** An API that enables your application to **recognize keywords or short phrases**.
- **Intent Recognition:** An API that uses **conversational language understanding to determine the semantic meaning of spoken input**.

This module focuses on speech recognition and speech synthesis, which are core capabilities of any speech-enabled application.

Note: The code examples in this module are provided in Python, but you can use any of the available Azure AI Speech SDK packages to develop speech-enabled applications in your preferred language. Available SDK packages include:

- [azure-cognitiveservices-speech for Python](#)
- [Microsoft.CognitiveServices.Speech for Microsoft .NET](#)
- [microsoft-cognitiveservices-speech-sdk for JavaScript](#)
- [Microsoft Cognitive Services Speech SDK For Java](#)

Provision an Azure resource for speech

Before you can use Azure AI Speech, you need to create an Azure AI Speech resource in your Azure subscription. You can use either **a dedicated Azure AI Speech resource** or **a multi-service Azure AI Services or Azure AI Foundry resource**.

After you create your resource, you'll need the following information to use it from a client application through one of the supported SDKs:

- The location in which the resource is deployed (for example, eastus)
- One of the keys assigned to your resource.

You can view of these values on the **Keys** and **Endpoint** page for your resource in the Azure portal.

While the specific syntax and parameters can vary between language-specific SDKs, most interactions with the Azure AI Speech service start with the creation of a `SpeechConfig` object that encapsulates the connection to your Azure AI Speech resource.

For example, the following Python code instantiates a `SpeechConfig` object based on an Azure AI Speech resource in the East US region:

```

import azure.cognitiveservices.speech as speech_sdk

speech_config = speech_sdk.SpeechConfig(your_project_key, 'eastus')

```

Note: This example assumes that the Speech SDK package for python has been installed, like this:

```
pip install azure-cognitiveservices-speech
```

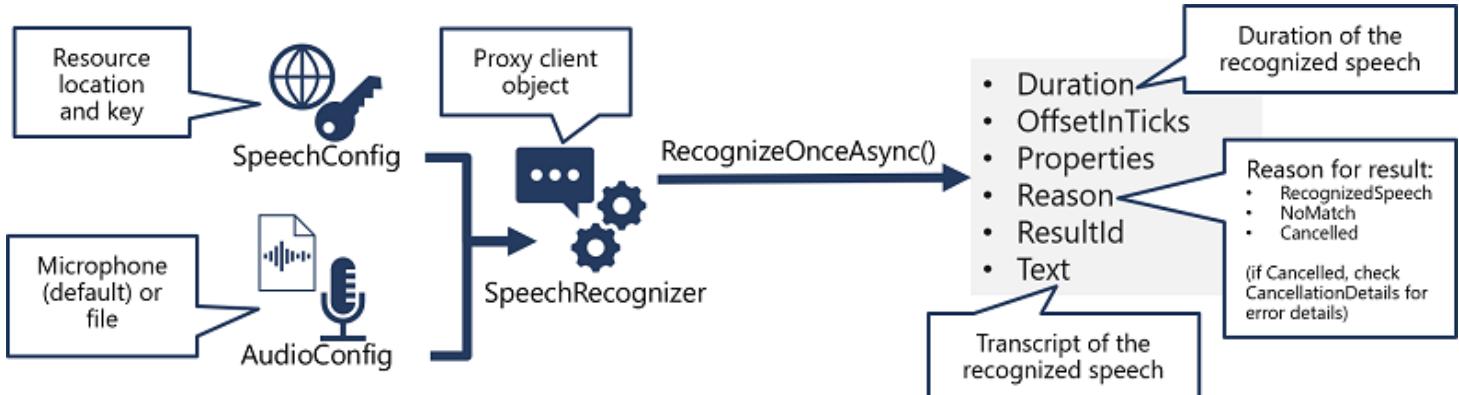
Use the Azure AI Speech to Text API

The Azure AI Speech service supports **speech recognition** through the following features:

- **Real-time transcription:** Instant transcription with intermediate results for live audio inputs.
- **Fast transcription:** Fastest synchronous output for situations with predictable latency.
- **Batch transcription:** Efficient processing for large volumes of prerecorded audio.
- **Custom speech:** Models with **enhanced accuracy for specific domains and conditions.**

Using the Azure AI Speech SDK

While the specific details vary, depending on the SDK being used (Python, C#, and so on); there's a consistent pattern for using the **Speech to text** API:



- Use a **SpeechConfig** object to encapsulate the information required to connect to your Azure AI Speech resource. Specifically, its **location** and **key**.
- Optionally, use an **AudioConfig** to define the input source for the audio to be transcribed. By default, this is the default system microphone, but you can also specify an audio file.
- Use the **SpeechConfig** and **AudioConfig** to create a **SpeechRecognizer** object. This object is a proxy client for the **Speech to text** API.

- Use the methods of the **SpeechRecognizer** object to call the underlying API functions. For example, the **RecognizeOnceAsync()** method uses the Azure AI Speech service to *asynchronously* transcribe a single spoken utterance.
- Process the response from the Azure AI Speech service. In the case of the **RecognizeOnceAsync()** method, the result is a **SpeechRecognitionResult** object that includes the following properties:
 - Duration
 - OffsetInTicks
 - Properties
 - Reason
 - ResultId
 - Text

If the operation was successful, the **Reason** property has the enumerated value **RecognizedSpeech**, and the **Text** property contains the transcription. Other possible values for **Result** include **NoMatch** (indicating that the audio was successfully parsed but no speech was recognized) or **Canceled**, indicating that an error occurred (in which case, you can check the **Properties** collection for the **CancellationReason** property to determine what went wrong).

Use the text to speech API

Similarly to its Speech to text APIs, the Azure AI Speech service offers other **REST APIs for speech synthesis**:

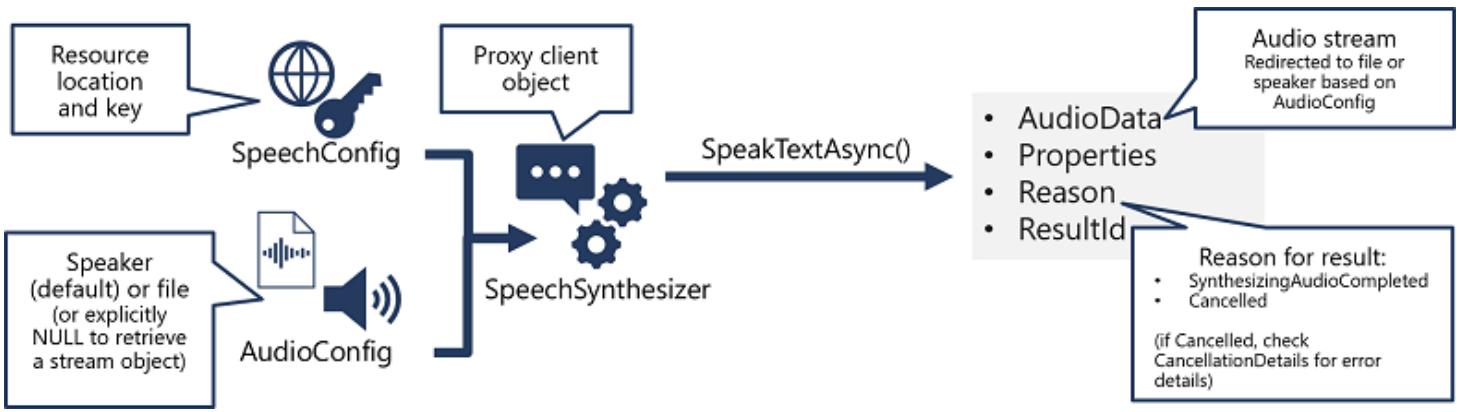
- The **Text to speech API**, which is the primary way to perform speech synthesis.
- The **Batch synthesis API**, which is designed to support batch operations that *convert large volumes of text to audio* - for example to *generate an audio-book from the source text*.

You can learn more about the REST APIs in the [Text to speech REST API documentation](#). In practice, most interactive speech-enabled applications use the Azure AI Speech service through a (programming) language-specific SDK.

Using the Azure AI Speech SDK

As with speech recognition, in practice most interactive speech-enabled applications are built using the Azure AI Speech SDK.

The pattern for implementing speech synthesis is similar to that of speech recognition:



- Use a `SpeechConfig` object to encapsulate the information required to connect to your Azure AI Speech resource. Specifically, its **location** and **key**.
- Optionally, use an `AudioConfig` to define the output device for the speech to be synthesized. By default, this is the default system speaker, but you can also specify an audio file, or by explicitly setting this value to a `null` value, you can process the audio stream object that is returned directly.
- Use the `SpeechConfig` and `AudioConfig` to create a `SpeechSynthesizer` object. This object is a proxy client for the **Text to speech API**.
- Use the methods of the `SpeechSynthesizer` object to call the underlying API functions. For example, the `SpeakTextAsync()` method uses the Azure AI Speech service to convert text to spoken audio.
- Process the response from the Azure AI Speech service. In the case of the `SpeakTextAsync` method, the result is a `SpeechSynthesisResult` object that contains the following properties:
 - `AudioData`
 - `Properties`
 - `Reason`
 - `ResultId`

When speech has been successfully synthesized, the `Reason` property is set to the `SynthesizingAudioCompleted` enumeration and the `AudioData` property contains the audio stream (which, depending on the `AudioConfig` may have been automatically sent to a speaker or file).

Configure audio format and voices

When synthesizing speech, you can use a `SpeechConfig` object to customize the audio that is returned by the Azure AI Speech service.

Audio format

The Azure AI Speech service supports multiple output formats for the audio stream that is generated by speech synthesis. Depending on your specific needs, you can choose a format based on the required:

- Audio file type
- Sample-rate
- Bit-depth

For example, the following Python code sets the speech output format for a previously defined `SpeechConfig` object named `speech_config`:

```
speech_config.set_speech_synthesis_output_format(SpeechSynthesisOutputFormat.Riff24Khz16BitMonoI
```

For a full list of supported formats and their enumeration values, see the [Azure AI Speech SDK documentation](#).

Voices

The Azure AI Speech service provides multiple voices that you can use to personalize your speech-enabled applications. Voices are **identified by names** that indicate a locale and a person's name - for example `en-GB-George`.

The following Python example code sets the voice to be used

```
speech_config.speech_synthesis_voice_name = "en-GB-George"
```

For information about voices, see the [Azure AI Speech SDK documentation](#).

Use Speech Synthesis Markup Language

While the Azure AI Speech SDK enables you to submit plain text to be synthesized into speech, the service also supports an XML-based syntax for describing characteristics of the speech you want to generate. This **Speech Synthesis Markup Language (SSML)** syntax offers greater control over how the spoken output sounds, enabling you to:

- Specify a **speaking style**, such as `"excited"` or `"cheerful"` when using a neural voice.

- Insert **pauses or silence**.
- Specify **phonemes** (phonetic pronunciations), for example to pronounce the text "SQL" as "sequel".
- Adjust the **prosody** of the voice (affecting the pitch, timbre, and speaking rate).
- Use **common "say-as" rules**, for example to specify that a given string should be expressed as a date, time, telephone number, or other form.
- Insert **recorded speech or audio**, for example to include a standard recorded message or simulate background noise.

For example, consider the following SSML:

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
        xmlns:mstts="https://www.w3.org/2001/mstts" xml:lang="en-US">
    <voice name="en-US-AriaNeural">
        <mstts:express-as style="cheerful">
            I say tomato
        </mstts:express-as>
    </voice>
    <voice name="en-US-GuyNeural">
        I say <phoneme alphabet="sapi" ph="t ao m ae t ow"> tomato </phoneme>.
        <break strength="weak"/> Lets call the whole thing off!
    </voice>
</speak>
```

This SSML specifies a spoken dialog between two different **neural voices**, like this:

- Ariana (cheerfully): "I say tomato:
- Guy: "I say tomato (pronounced tom-ah-toe) ... Let's call the whole thing off!"

To submit an SSML description to the Speech service, you can use an appropriate method of a `SpeechSynthesizer` object, like this:

```
speech_synthesizer.speak_ssml('<speak>...');
```

For more information about SSML, see the [Azure AI Speech SDK documentation](#).

Exercise - Create a speech-enabled app

In this exercise, build a speech enabled app for both speech recognition and synthesis.

Recognize and synthesize speech

Azure AI Speech is a service that provides speech-related functionality, including:

- A *speech-to-text API* that enables you to implement *speech recognition* (converting audible spoken words into text).
- A *text-to-speech API* that enables you to implement *speech synthesis* (converting text into audible speech).

In this exercise, you'll use both of these APIs to implement a speaking clock application.

Module assessment

1. What information do you need from your Azure AI Speech service resource to consume it using the Azure AI Speech SDK? **The location and one of the keys.**
NOTE: Location is not endpoint!! An example of location: eastus
2. Which object should you use to specify that the speech input to be transcribed to text is in an audio file? **AudioConfig** [Use the Azure AI Speech to Text API](#)
3. How can you change the voice used in speech synthesis? Set the `SpeechSynthesisVoiceName` property of the `SpeechConfig` object to the desired voice name.

Summary

In this module, you learned how to:

- Provision an Azure resource for the Azure AI Speech service
- Use the **Speech to text API** to implement **speech recognition**
- Use the **Text to speech API** to implement **speech synthesis**
- Configure audio format and voices
- Use **Speech Synthesis Markup Language (SSML)**

To learn more about the Azure AI Speech, refer to the [Azure AI Speech service documentation](#).

Translate speech with the Azure AI Speech service

Translation of speech builds on speech recognition by recognizing and transcribing spoken input in a specified language, and returning translations of the transcription in one or more other languages.

Learning objectives

In this module, you will learn how to:

- Provision Azure resources for **speech translation**.
- Generate text translation from speech.
- Synthesize spoken translations.

Introduction

Translation of speech builds on speech recognition by recognizing and transcribing spoken input in a specified language, and returning translations of the transcription in one or more other languages.

In this module, you'll learn how to:

- Provision Azure resources for speech translation.
- Generate text translation from speech.
- Synthesize spoken translations.

The units in the module include important conceptual information about Azure AI Speech and how to use its API through one of the supported software development kits (SDKs), after which you're able to try Azure AI Speech for yourself in a hands-on exercise. To complete the hands-on exercise, you'll need a Microsoft Azure subscription. If you don't already have one, you can sign up for a [free trial](#).

Provision an Azure resource for speech translation

The Azure AI Speech service provides robust, machine learning and artificial intelligence-based speech translation services, enabling developers to add end-to-end, real-time, speech translations to their applications or services. You can use either **a dedicated Azure AI Speech resource** or **a multi-service Azure AI Services resource**.

Before you can use the service, you need to create an Azure AI Speech resource in your Azure subscription.

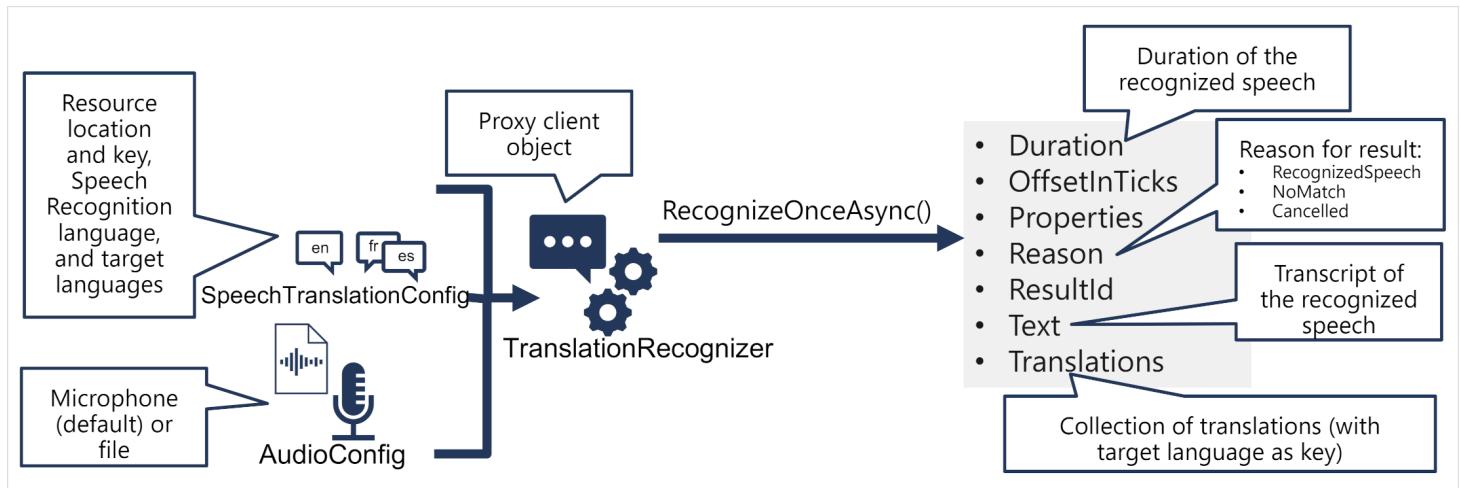
After creating your Azure resource, you'll need the following information to use it from a client application through one of the supported SDKs:

- The **location** in which the resource is deployed (for example, eastus)
- **One of the keys** assigned to your resource.

You can view of these values on the **Keys and Endpoint page** for your resource in the Azure portal.

Translate speech to text

The pattern for **speech translation** using the Azure AI Speech SDK is **similar to speech recognition**, with the addition of information about the source and target languages for translation:



- Use a `SpeechTranslationConfig` object to encapsulate the information required to connect to your Azure AI Speech resource. Specifically, its **location and key**.
- The `SpeechTranslationConfig` object is also used to specify the **speech recognition language** (the language in which the input speech is spoken) and the **target languages** into which it should

be translated.

- Optionally, use an `AudioConfig` to define the input source for the audio to be transcribed. By default, this is the **default system microphone**, but you can also specify an **audio file**.
- Use the `SpeechTranslationConfig`, and `AudioConfig` to create a `TranslationRecognizer` object. This object is a **proxy client for the Azure AI Speech translation API**.
- Use the methods of the `TranslationRecognizer` object to call the underlying API functions. For example, the `RecognizeOnceAsync()` method uses the Azure AI Speech service to **asynchronously** translate a single spoken utterance.
- Process the response from Azure AI Speech. In the case of the `RecognizeOnceAsync()` method, the result is a `SpeechRecognitionResult` object that includes the following properties:
 - Duration
 - OffsetInTicks
 - Properties
 - Reason
 - ResultId
 - Text
 - Translations

If the operation was successful, the `Reason` property has the enumerated value `RecognizedSpeech`, the `Text` property contains the transcription in the original language. You can also access a `Translations` property which contains a dictionary of the translations (using the two-character ISO language code, such as "en" for English, as a key).

Synthesize translations

The `TranslationRecognizer` returns *translated transcriptions of spoken input* - essentially *translating audible speech to text*.

You can also synthesize the translation as speech to create **speech-to-speech translation** solutions. There are two ways you can accomplish this.

Event-based synthesis

When you want to perform **1:1 translation (translating from one source language into a single target language)**, you can use event-based synthesis to capture the translation as an audio stream. To do this, you need to:

Specify the desired voice for the translated speech in the `TranslationConfig`. Create an event handler for the `TranslationRecognizer` object's `Synthesizing` event. In the event handler, use the `GetAudio()` method of the `Result` parameter to retrieve the byte stream of translated audio. The specific code used to implement an event handler varies depending on the programming language you're using. See the [C#](#) and [Python](#) examples in the Speech SDK documentation.

Manual synthesis

Manual synthesis is an alternative approach to event-based synthesis that doesn't require you to implement an event handler. You can use manual synthesis to generate audio translations for one or more target languages.

Manual synthesis of translations is essentially just the combination of two separate operations in which you:

- Use a `TranslationRecognizer` to translate spoken input into text transcriptions in one or more target languages.
- Iterate through the `Translations` dictionary in the result of the translation operation, using a `SpeechSynthesizer` to synthesize an audio stream for each language.

Exercise - Translate speech

In this exercise, build an app that recognizes and translates speech into a specific language.

Translate Speech

Azure AI Speech includes a speech translation API that you can use to translate spoken language. For example, suppose you want to develop a translator application that people can use when traveling in places where they don't speak the local language. They would be able to say phrases such as "Where is the station?" or "I need to find a pharmacy" in their own language, and have it translate them to the local language.

Module assessment

1. Which SDK object should you use to specify the language(s) into which you want speech translated? **SpeechTranslationConfig**
2. Which SDK object should you use as a proxy for the Translation API of Azure AI Speech service? **TranslationRecognizer**
3. When translating speech, in which cases can you use the Synthesizing event to synthesize the translations and speech? **Only when translating to a single target language.**

Summary

Now that you've completed this module, you learned how to:

- Provision Azure resources for speech translation.
- Generate text translation from speech.
- Synthesize spoken translations.

For more information about speech translation, refer to the [Azure AI Speech documentation](#).

Develop an audio-enabled generative AI application

A voice carries meaning beyond words, and audio-enabled generative AI models can interpret spoken input to **understand tone, intent, and language**. Learn how to build **audio-enabled chat apps** that listen and respond to audio.

Learning objectives

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

- Deploy an **audio-enabled generative AI model** in Azure AI Foundry.
- Create a **chat app** that submits audio-based prompts.

Introduction

Generative AI models make it possible to build intelligent chat-based applications that can understand and reason over input. Traditionally, text input is the primary mode of interaction with AI models, but **multimodal models** are increasingly becoming available. These models make it possible for chat applications to respond to audio input as well as text.

In this module, we'll discuss **audio-enabled generative AI** and explore how you can use Azure AI Foundry to create generative AI solutions that respond to prompts that include a mix of text and audio data.

Deploy a multimodal model

To handle prompts that include audio, you need to deploy a **multimodal generative AI model** - in other words, a model that supports not only *text-based input*, but *audio-based input* as well.

Multimodal models available in Azure AI Foundry include (among others):

- Microsoft Phi-4-multimodal-instruct

- OpenAI gpt-4o
- OpenAI gpt-4o-mini

Tip: To learn more about available models in Azure AI Foundry, see the [Model catalog and collections in Azure AI Foundry portal](#) article in the Azure AI Foundry documentation.

Testing multimodal models with audio-based prompts

After deploying a multimodal model, you can test it in the chat playground in Azure AI Foundry portal. Some models allow you to include audio attachments in the playground, either by **uploading a file or recording a message**.

← Chat playground ▾

The screenshot shows the Azure AI Foundry Chat playground interface. At the top, there are navigation links: View code, Evaluate, Deploy, Import, and Export. On the left, under the 'Setup' tab, there's a 'Deployment' section showing 'gpt-4o-audio-preview (version:2024-12-17)'. Below it, a text input area contains the instruction: 'You are an AI assistant that helps people find information.' There are buttons for 'Apply changes', 'Generate prompt', and '+ Add section'. Under 'Choose a voice', the selected voice is 'Alloy'. Below that are sections for 'Add your data' (with a 'PREVIEW' button) and 'Parameters'. On the right, the 'Chat history' pane shows a message from the user: 'Can you transcribe this audio file?'. Below it is an audio file card with a duration of '00:03' and three dots. A transcription of the audio is shown: 'The audio says, "Me gustaría comprar dos manzanas," which translates to "I would like to buy two apples" in English.' At the bottom of the chat history, there's a text input field with placeholder text 'Type user query here. (Shift + Enter for new line)' and a row of buttons: a file icon, a microphone icon, a 'Record' button (which is currently selected), and a 'Send' button.

In the chat playground, you can upload a local audio file and add text to the message to elicit a response from a multimodal model.

Develop an audio-based chat app

To develop a client app that engages in audio-based chats with a multimodal model, you can use the same basic techniques used for text-based chats. You require a connection to the **endpoint** where the model is deployed, and you use that endpoint to submit prompts that consists of messages to the model and process the responses.

The key difference is that **prompts for an audio-based chat** include multi-part user messages that contain **both a text content item and an audio content item**.

The JSON representation of a prompt that includes a multi-part user message looks something like this:

```
{
  "messages": [
    { "role": "system", "content": "You are a helpful assistant." },
    { "role": "user", "content": [
      {
        "type": "text",
        "text": "Transcribe this audio:"
      },
      {
        "type": "audio_url",
        "audio_url": {
          "url": "https://....."
        }
      }
    ] }
  ]
}
```

The audio content item can be:

- **A URL to an audio file in a web site.**
- **Binary audio data.**

When using binary data to submit a local audio file, the `audio_url` content takes the form of a base64 encoded value in a data URL format:

```
{  
  "type": "audio_url",  
  "audio_url": {  
    "url": "data:audio/mp3;base64,<binary_audio_data>"  
  }  
}
```

Depending on the model type, and where you deployed it, you can **use Microsoft Azure AI Model Inference or OpenAI APIs to submit audio-based prompts**. These libraries also provide language-specific SDKs that abstract the underlying REST APIs.

In the exercise that follows in this module, you can *use the Python or .NET SDK for the Azure AI Model Inference API and the OpenAI API to develop an audio-enabled chat application.*

Exercise - Develop an audio-enabled chat app

If you have an Azure subscription, you can complete this exercise to develop an audio-enabled chat app.

Develop an audio-enabled chat app

In this exercise, you use the `Phi-4-multimodal-instruct` generative AI model to generate responses to prompts that include audio files. You'll develop an app that provides AI assistance for a produce supplier company by using Azure AI Foundry and the Azure AI Model Inference service to summarize voice messages left by customers.

Module assessment

1. Which kind of model can you use to respond to audio input? **Multimodal models**.
2. How can you submit a prompt that asks a model to analyze an audio file? **Submit a prompt that contains a multi-part user message, containing both text content and audio content.**
3. How can you include an audio in a message? **As a URL or as binary data.**

Summary

In this module, you learned about **audio-enabled generative AI models** and how to implement chat solutions that include audio-based input.

Audio-enabled models let you create AI solutions that can understand audio and respond to related questions or instructions. Beyond just identifying spoken words, some models can also use reasoning based on what they hear. For instance, they can **summarize a message or assess the speaker's sentiment**.

Tip: For more information about working with multimodal models in Azure AI Foundry, see [How to use image and audio in chat completions with Azure AI model inference](#) and [Quickstart: Use speech and audio in your AI chats](#).

Develop an Azure AI Voice Live agent

Learn how to develop an Azure AI Voice Live agent using the Voice Live API and SDK. This module covers the fundamentals of the Voice Live platform, including API integration, SDK usage, and building conversational AI agents.

Learning objectives

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

- Implement the Azure AI Voice Live API to enable real-time, bidirectional communication.
- Set up and configure the agent session.
- Develop and manage event handlers to create dynamic and interactive user experiences.
- Build and deploy a Python-based web app with real-time voice interaction capabilities to Azure.

Introduction

Voice-enabled applications are transforming how we interact with technology, and this module guides you through building a real-time, interactive voice solutions using advanced APIs and tools. The Azure AI Voice live API is a solution enabling low-latency, high-quality speech to speech interactions for voice agents. The API is designed for developers seeking scalable and efficient voice-driven experiences as it eliminates the need to manually orchestrate multiple components.

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

- Implement the Azure AI Voice Live API to enable real-time, bidirectional communication.
- Set up and configure the agent session.
- Develop and manage event handlers to create dynamic and interactive user experiences.
- Build and deploy a Python-based web app with real-time voice interaction capabilities to Azure.

Explore the Azure Voice Live API

The Voice live API enables developers to create voice-enabled applications with real-time, bidirectional communication. This unit explores its architecture, configuration, and implementation.

Key features of the Voice Live API

The Voice live API provides real-time communication using WebSocket connections. It supports advanced features such as speech recognition, text-to-speech synthesis, avatar streaming, and audio processing.

- JSON-formatted events manage conversations, audio streams, and responses.
- Events are categorized into client events (sent from client to server) and server events (sent from server to client).

Key features include:

- Real-time audio processing with support for multiple formats like PCM16 and G.711.
- Advanced voice options, including OpenAI voices and Azure custom voices.
- Avatar integration using WebRTC for video and animation.
- Built-in noise reduction and echo cancellation.

Note: Voice Live API is optimized for Azure AI Foundry resources. We recommend using Azure AI Foundry resources for full feature availability and best Azure AI Foundry integration experience.

For a table of supported models and regions, visit [the Voice Live API overview](#).

Connect to the Voice Live API

The Voice live API supports two authentication methods: Microsoft Entra (keyless) and API key. Microsoft Entra uses token-based authentication for an Azure AI Foundry resource. You apply a retrieved authentication token using a Bearer token with the Authorization header.

For the recommended keyless authentication with Microsoft Entra ID, you need to assign the Cognitive Services User role to your user account or a managed identity. You generate a token using the Azure CLI or Azure SDKs. The token must be generated with the <https://ai.azure.com/.default> scope, or the legacy <https://cognitiveservices.azure.com/.default> scope. Use the token in the Authorization header of the WebSocket connection request, with the format `Bearer <token>`.

For key access, an API key can be provided in one of two ways. You can use an api-key connection header on the prehandshake connection. This option isn't available in a browser environment. Or, you can use an api-key query string parameter on the request URI. Query string parameters are encrypted when using https/wss.

Note: The `api-key` connection header on the prehandshake connection isn't available in a browser environment.

WebSocket endpoint

The endpoint to use varies depending on how you want to access your resources. You can access resources through a connection to the AI Foundry project (Agent), or through a connection to the model.

- Project connection: The endpoint is

`wss://<your-ai-foundry-resource-name>.services.ai.azure.com/voice-live/realtim?api-version=2025-10-01`

- Model connection: The endpoint is

`wss://<your-ai-foundry-resource-name>.cognitiveservices.azure.com/voice-live/realtim?api-version=2025-10-01`

The endpoint is the same for all models. The only difference is the required model query parameter, or, when using the Agent service, the `agent_id` and `project_id` parameters.

Voice Live API events

Client and server events facilitate communication and control within the Voice live API. Key client events include:

- `session.update` : Modify session configurations.
- `input_audio_buffer.append` : Add audio data to the buffer.
- `response.create` : Generate responses via model inference.

Server events provide feedback and status updates:

- `session.updated` : Confirm session configuration changes.
- `response.done` : Indicate response generation completion.
- `conversation.item.created` : Notify when a new conversation item is added.

For a full list of client/server events, visit [Voice live API Reference](#).

Note: Proper handling of events ensures seamless interaction between client and server.

Configure session settings for the Voice live API

Often, the first event sent by the caller on a newly established Voice live API session is the `session.update` event. This event controls a wide set of input and output behavior. Session settings can be updated dynamically using the `session.update` event. Developers can configure voice types, modalities, turn detection, and audio formats.

Example configuration:

```
{
  "type": "session.update",
  "session": {
    "modalities": ["text", "audio"],
    "voice": {
      "type": "openai",
      "name": "alloy"
    },
    "instructions": "You are a helpful assistant. Be concise and friendly.",
    "input_audio_format": "pcm16",
    "output_audio_format": "pcm16",
    "input_audio_sampling_rate": 24000,
    "turn_detection": {
      "type": "azure_semantic_vad",
      "threshold": 0.5,
      "prefix_padding_ms": 300,
      "silence_duration_ms": 500
    },
    "temperature": 0.8,
    "max_response_output_tokens": "inf"
  }
}
```

Tip: Use Azure semantic VAD for intelligent turn detection and improved conversational flow.

Implement real-time audio processing with the Voice live API

Real-time audio processing is a core feature of the Voice live API. Developers can append, commit, and clear audio buffers using specific client events.

- Append audio: Add audio bytes to the input buffer.
- Commit audio: Process the audio buffer for transcription or response generation.
- Clear audio: Remove audio data from the buffer.

Noise reduction and echo cancellation can be configured to enhance audio quality. For example:

```
{  
  "type": "session.update",  
  "session": {  
    "input_audio_noise_reduction": {  
      "type": "azure_deep_noise_suppression"  
    },  
    "input_audio_echo_cancellation": {  
      "type": "server_echo_cancellation"  
    }  
  }  
}
```

Note: Noise reduction improves VAD accuracy and model performance by filtering input audio.

Integrate avatar streaming using the Voice live API

The Voice live API supports WebRTC-based avatar streaming for interactive applications. Developers can configure video, animation, and blendshape settings.

- Use the session.avatar.connect event to provide the client's SDP offer.
- Configure video resolution, bitrate, and codec settings.
- Define animation outputs such as blendshapes and visemes.

Example configuration:

```
{  
  "type": "session.avatar.connect",  
  "client_sdp": "<client_sdp>"  
}
```

Tip: Use high-resolution video settings for enhanced visual quality in avatar interactions.

Explore the AI Voice Live client library for Python

The Azure AI Voice Live client library for Python provides a real-time, speech-to-speech client for Azure AI Voice Live API. It opens a WebSocket session to stream microphone audio to the service and receives server events for responsive conversations.

Important: As of version 1.0.0, this SDK is async-only. The synchronous API is deprecated to focus exclusively on async patterns. All examples and samples use `async/await` syntax.

In this unit, you learn how to use the SDK to implement authentication and handle events. You also see a minimal example of creating a session. For a full reference to the Voice Live package, visit the [voice live Package reference](#).

Implement authentication

You can implement authentication with an API key or a Microsoft Entra ID token. The following code sample shows an API key implementation. It assumes environment variables are set in a `.env` file, or directly in your environment.

```
import asyncio
from azure.core.credentials import AzureKeyCredential
from azure.ai.vovcelive import connect

async def main():
    async with connect(
        endpoint="your-endpoint",
        credential=AzureKeyCredential("your-api-key"),
        model="gpt-4o"
    ) as connection:
        # Your async code here
        pass

asyncio.run(main())
```

For production applications, Microsoft Entra authentication is recommended. The following code sample shows implementing the `DefaultAzureCredential` for authentication:

```

import asyncio
from azure.identity.aio import DefaultAzureCredential
from azure.ai.voicelive import connect

async def main():
    credential = DefaultAzureCredential()

    async with connect(
        endpoint="your-endpoint",
        credential=credential,
        model="gpt-4o"
    ) as connection:
        # Your async code here
        pass

asyncio.run(main())

```

Handling events

Proper handling of events ensures a more seamless interaction between the client and agent. For example, when handling a user interrupting the voice agent you need to cancel agent audio playback immediately in the client. If you don't, the client continues to play the last agent response until the interrupt is processed in the API - resulting in the agent "talking over" the user.

The following code sample shows some basic event handling:

```

async for event in connection:
    if event.type == ServerEventType.SESSION_UPDATED:
        print(f"Session ready: {event.session.id}")
        # Start audio capture

    elif event.type == ServerEventType.INPUT_AUDIO_BUFFER_SPEECH_STARTED:
        print("User started speaking")
        # Stop playback and cancel any current response

    elif event.type == ServerEventType.RESPONSE_AUDIO_DELTA:
        # Play the audio chunk
        audio_bytes = event.delta

    elif event.type == ServerEventType.ERROR:
        print(f"Error: {event.error.message}")

```

Minimal example

The following code sample shows authenticating to the API and configuring the session.

```

import asyncio
from azure.core.credentials import AzureKeyCredential
from azure.ai.vovcelive.aio import connect
from azure.ai.vovcelive.models import (
    RequestSession, Modality, InputAudioFormat, OutputAudioFormat, ServerVad, ServerEventType
)

API_KEY = "your-api-key"
ENDPOINT = "your-endpoint"
MODEL = "gpt-4o"

async def main():
    async with connect(
        endpoint=ENDPOINT,
        credential=AzureKeyCredential(API_KEY),
        model=MODEL,
    ) as conn:
        session = RequestSession(
            modalities=[Modality.TEXT, Modality.AUDIO],
            instructions="You are a helpful assistant.",
            input_audio_format=InputAudioFormat.PCM16,
            output_audio_format=OutputAudioFormat.PCM16,
            turn_detection=ServerVad(
                threshold=0.5,
                prefix_padding_ms=300,
                silence_duration_ms=500
            ),
        )
        await conn.session.update(session=session)

    # Process events
    async for evt in conn:
        print(f"Event: {evt.type}")
        if evt.type == ServerEventType.RESPONSE_DONE:
            break

asyncio.run(main())

```

Exercise - Develop an Azure AI Voice Live agent

In this exercise, you complete a Flask-based Python web app based that enables real-time voice interactions with an agent. You add the code to initialize the session, and handle session events. You use a deployment script that: deploys the AI model; creates an image of the app in Azure Container Registry (ACR) using ACR tasks; and then creates an Azure App Service instance that pulls the image. To test the app, you need an audio device with microphone and speaker capabilities.

While this exercise is based on Python, you can develop similar applications other language-specific SDKs; including:

- Azure VoiceLive client library for .NET

Tasks performed in this exercise:

- Download the base files for the app
- Add code to complete the web app
- Review the overall code base
- Update and run the deployment script
- View and test the application

This exercise takes approximately 30 minutes to complete.

Before you start

To complete the exercise, you need:

- An Azure subscription. If you don't already have one, you can sign up for one <https://azure.microsoft.com/>.
- An audio device with microphone and speaker capabilities.

Get started

Select the Launch Exercise button to open the exercise instructions in a new browser window. When you're finished with the exercise, return here to:

- Complete the module
- Earn a badge for completing this module

Develop an Azure AI Voice Live voice agent

In this exercise, you complete a Flask-based Python web app based that enables real-time voice interactions with an agent. You add the code to initialize the session, and handle session events. You use a deployment script that: deploys the AI model; creates an image of the app in Azure Container Registry (ACR) using ACR tasks; and then creates an Azure App Service instance that pulls the the image. To test the app you will need an audio device with microphone and speaker capabilities.

Module assessment

1. What are the two authentication methods supported by the Voice Live API? Microsoft Entra (keyless) and API key
2. Which scope is required when generating a token for Microsoft Entra authentication?
<https://cognitiveservices.azure.com/.default>
3. Which protocol is used for avatar streaming integration in Voice Live API? WebRTC

4. Which event should be handled to stop audio playback when a user interrupts the voice agent?
ServerEventType.INPUT_AUDIO_BUFFER_SPEECH_STARTED
5. What is the recommended authentication method for production applications using the SDK? Microsoft Entra authentication with DefaultAzureCredential

Summary

In this module, you learned about the Voice live API's features, including WebSocket connections, speech recognition, text-to-speech synthesis, and avatar streaming. You also explored Azure AI Voice Live for creating real-time speech-to-speech applications using Python, including setting up the client library and managing sessions. Additionally, you learned how to implement event handlers in Python for dynamic responses and real-time audio processing. Finally, you developed a Python-based web application using Flask, integrated it with Azure resources, and tested the application.

Additional reading

- [What is the Speech service?](#)
- [How to customize voice live input and output](#)

Analyze images

With the Azure AI Vision service, you can use **pre-trained models** to **analyze images** and **extract insights and information** from them.

Learning objectives

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

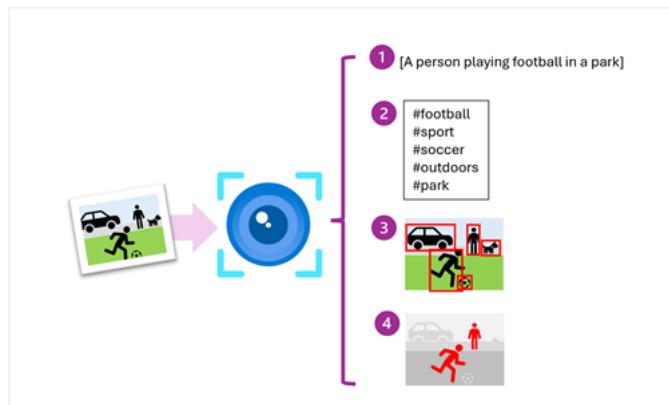
- Provision an Azure AI Vision resource.
- Use the Azure AI Vision SDK to connect to your resource.
- Write code to analyze an image.

Introduction

Computer Vision is a branch of artificial intelligence (AI) in which software **interprets visual input, often from images or video feeds**. In Microsoft Azure, you can use the Azure AI Vision service to implement multiple computer vision scenarios, including:

- *Image analysis*
- *Optical character recognition (OCR)*
- *Face detection and analysis*
- *Video analysis*

In this module, we'll focus on **image analysis**, and explore how to build applications that use the Azure AI Vision service to **analyze and extract and infer insights from images**.

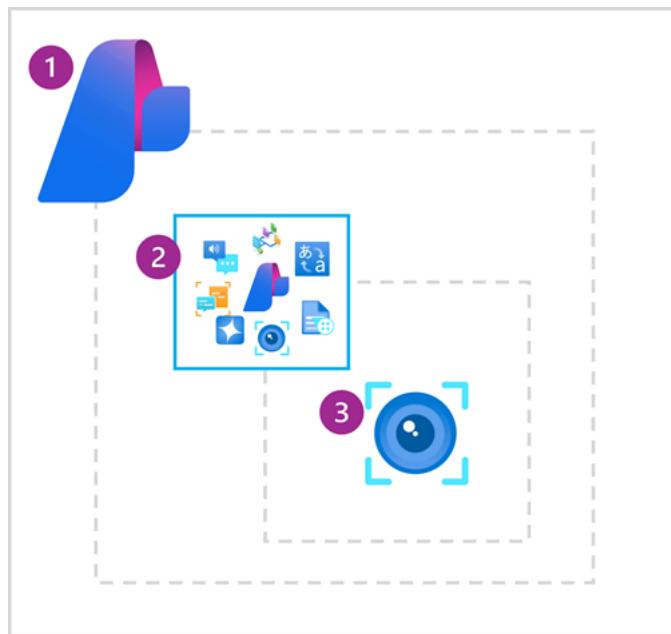


As shown in this conceptual diagram, the Azure AI Vision service provides services that you can use to analyze images and:

- Generate a **caption** for an image based on its contents.
- Suggest appropriate **tags** to associate with an image.
- **Detect and locate common objects** in an image.
- **Detect and locate people** in an image.

Provision an Azure AI Vision resource

To use Azure AI Vision image analysis services, you need to provision an **Azure AI Vision resource** in your Azure subscription. You can choose from multiple provisioning options:



1. Create an **Azure AI Foundry** project and an associated **hub**. By default, an **Azure AI Foundry hub** includes an **Azure AI services** multi-service resource, which includes Azure AI Vision. Azure AI Foundry projects are recommended for development of AI solutions on Azure that combine generative AI, agents, and pre-built Azure AI services, or which involve collaborative development by a team of software engineers and service operators.
2. If you don't need all of the functionality in an Azure AI Foundry hub, you can create an **Azure AI services multi-service resource** in your Azure subscription. You can then use this resource to access Azure AI Vision services and other AI services through a **single endpoint** and **key**.
3. If you only need to use Azure AI Vision functionality, or you're just experimenting with the service, you can create a standalone **Computer Vision** resource in your Azure subscription. One benefit of this approach is that the standalone service provides a free tier that you can use to explore the service at no cost.

Tip: If you're unfamiliar with Azure AI Foundry and Azure AI services, consider completing the [Plan and prepare to develop AI solutions on Azure](#) module.

Connecting to your resource

After you've deployed your resource, you can use the [Azure AI Vision REST API](#) or a **language-specific SDK** (such as the Python SDK or Microsoft .NET SDK) to connect to it from a client application.

Every Azure AI Vision resource provides an **endpoint** to which client applications must connect. You can find the endpoint for your resource in the Azure portal, or if you're working in an Azure AI Foundry project, in the Azure AI Foundry portal. The endpoint is in the form of a URL, and typically looks something like this:

`https://<resource_name>.cognitiveservices.azure.com/`

To connect to the endpoint, client applications must be *authenticated*. Options for **authentication** include:

- **Key-based authentication:** Client applications are authenticated by passing an **authorization key** (which you can find and manage in the portal).
- **Microsoft Entra ID authentication:** Client applications are authenticated by using a **Microsoft Entra ID token** for credentials that have permission to access the Azure AI Vision resource in Azure.

When developing and testing an application, it's common to use key-based authentication or Microsoft Entra ID authentication based on your own Azure credentials. **In production, consider using Microsoft Entra ID authentication based on a managed identity for your Azure application or use Azure Key Vault to store authorization keys securely.**

Note: When using an Azure AI services resource in an Azure AI Foundry project, you can use the Azure AI Foundry SDK to connect to the project using Microsoft Entra ID authentication, and then retrieve the connection information for your Azure AI services resource, including the authorization key, from the project.

Analyze an image

After connecting to your Azure AI Vision resource endpoint, your client application can use the service to perform image analysis tasks.

Note the following requirements for image analysis:

- The image must be presented in **JPEG, PNG, GIF, or BMP** format.
- The file size of the image must be **less than 4 megabytes (MB)**.
- The dimensions of the image must be **greater than 50 x 50 pixels**.

Submitting an image for analysis

To analyze an image, you can use the [Analyze Image](#) REST method or the equivalent method in the SDK for your preferred programming language, specifying the visual features you want to include in the analysis.

```
from azure.ai.vision.imageanalysis import ImageAnalysisClient
from azure.ai.vision.imageanalysis.models import VisualFeatures
from azure.core.credentials import AzureKeyCredential

client = ImageAnalysisClient(endpoint=<YOUR_RESOURCE_ENDPOINT>, credential=AzureKeyCredential(<YOUR_AUTHORIZATION_KEY>) )

result = client.analyze(image_data=<IMAGE_DATA_BYTES>, # Binary data from your image file visual_features=[VisualFeatures.CAPTION, VisualF
```

Note: In this code example, the client app uses **key-based authentication**. To use Microsoft Entra ID authentication, you can use a **TokenCredential** instead of an **AzureKeyCredential**. The code example submits the image data as a binary object (which would typically be read from an image file). You can also analyze an image based on a URL by using the `analyze_from_url` method .

Available visual features are contained in the `VisualFeatures` enumeration:

- `VisualFeatures.Tags` : Identifies **tags** about the image, including objects, scenery, setting, and actions
- `VisualFeatures.Objects` : Returns the **bounding box** for each detected object
- `VisualFeatures.Caption` : Generates a **caption** of the image in natural language
- `VisualFeatures.DenseCaptions` : Generates **more detailed captions** for the objects detected
- `VisualFeatures.People` : Returns the **bounding box for detected people**
- `VisualFeatures.SmartCrops` : Returns the **bounding box of the specified aspect ratio** for the area of interest
- `VisualFeatures.Read` : Extracts **readable text**

Specifying the visual features you want analyzed in the image determines what information the response will contain. Most responses will contain a **bounding box** (if a location in the image is reasonable) or a **confidence score** (for features such as **tags or captions**).

Processing the response

This method returns a JSON document containing the requested information. The JSON response for image analysis looks similar to this example, depending on your requested features:

```
{ "apim-request-id": "abcde-1234-5678-9012-f1g2h3i4j5k6", "modelVersion": "<version>", "denseCaptionsResult": { "values": [ {
```

Exercise - Analyze images

In this exercise, you use the Azure AI Vision SDK to develop a client application that analyzes images.

Analyze images

Azure AI Vision is an artificial intelligence capability that enables software systems to interpret visual input by analyzing images. In Microsoft Azure, the **Vision** Azure AI service provides pre-built models for common computer vision tasks, including analysis of images to suggest captions and tags, detection of common objects and people. You can also use the Azure AI Vision service to remove the background or create a foreground matting of images.

Module assessment

1. What is the purpose of Azure AI Vision Image Analysis? **To extract information about visual features in images.**
2. You want to use Azure AI Vision Image Analysis to generate suggested text descriptions for an image. Which visual feature should you specify?
DenseCaptions

Summary

In this module, you learned how to provision an Azure AI Vision resource and use it from a client application to analyze images.

You can use Azure AI Vision's image analysis capabilities in scenarios that require information extraction or inference from images. A common use case is **digital asset management (DAM)**, in which you need to *tag, catalog, and index image-based data*.

To learn more about image analysis with the Azure AI Vision service, see the [Azure AI Vision documentation](#).

Read text in images with OCR

The **Azure AI Vision Image Analysis** service uses algorithms to process images and return information. This module teaches you how to use the Image Analysis API for **optical character recognition (OCR)**.

Learning objectives

In this module, you'll learn how to:

- Describe the OCR capabilities of **Azure AI Vision's Image Analysis API**.
- Use the Azure AI Vision service Image Analysis API to **extract text from images**.

Introduction

We live in a digital world, in which data is increasingly captured as images. Often, those images contain text, which you need to be able to extract from their pixelated format in the image for processing, indexing, and other tasks. Everyday examples include:

- Meeting a new business associate and taking a photograph of their **business card** to store their contact details digitally.
- **Scanning a document or ID card** to include in an application for a government or commercial service.
- Taking a photo of a **menu or recipe** to store it in a digital notebook.
- Photographing **street signs or store fronts** so you can submit the text they contain to a translation app.
- Digitizing **handwritten notes** using a cellphone camera.

In this module, we'll explore the **optical character recognition (OCR)** capabilities of the Azure AI Vision Image Analysis API, which makes these scenarios, and more, possible.

Explore Azure AI options for reading text

There are multiple Azure AI services that read text from documents and images, each optimized for results depending on the input and the specific requirements of your application.

Azure AI Vision

Azure AI Vision includes an *image analysis* capability that supports **optical character recognition (OCR)**. Consider using Azure AI Vision in the following scenarios:

- **Text location and extraction from scanned documents:** Azure AI Vision is a great solution for general, unstructured documents that have been scanned as images. For example, **reading text in labels, menus, or business cards**.
- **Finding and reading text in photographs:** Examples include photo's that include **street signs and store names**.
- **Digital asset management (DAM):** Azure AI Vision includes functionality for analyzing images beyond extracting text; including **object detection, describing or categorizing an image, generating smart-cropped thumbnails** and more. These capabilities make it a useful service when you need to catalog, index, or analyze large volumes of digital image-based content.

Azure AI Document Intelligence

Azure AI Document Intelligence is a service that you can use to **extract information from complex digital documents**. Azure AI Document Intelligence is designed for **extracting text, key-value pairs, tables, and structures** from documents automatically and accurately. Key considerations for choosing Azure AI Document Intelligence include:

- **Form processing:** Azure AI Document Intelligence is specifically designed to **extract data from forms, invoices, receipts, and other structured documents**.
- **Prebuilt models:** Azure AI Document Intelligence provides prebuilt models for common document types to reduce complexity and integrate into workflows or applications.

- **Custom models:** Creating custom models tailored to your specific documents, makes Azure AI Document Intelligence a flexible solution that can be used in many business scenarios.

Azure AI Content Understanding

Azure AI Content Understanding is a service that you can use to *analyze and extract information from multiple kinds of content*; including **documents, images, audio streams, and video**. It is suitable for:

- **Multimodal content extraction:** **Extracting content and structured fields** from documents, forms, audio, video, and images.
- **Custom content analysis scenarios:** Support for customizable analyzers enables you to extract specific content or fields tailored to business needs.

Note: In the rest of this module, we'll focus on the OCR image analysis feature in Azure AI Vision. To learn more about Azure AI Document Intelligence and Azure AI Content understanding, consider completing the following training modules:

[Plan an Azure AI Document Intelligence solution](#)

[Analyze content with Azure AI Content Understanding](#)

Read text with Azure AI Vision Image Analysis

To use Azure AI Vision for image analysis, including optical character recognition, you must provision an **Azure AI Vision resource** in an Azure subscription. The resource can be:

- An **Azure AI Services** multi-service resource (either deployed as part of an Azure AI Foundry hub and project, or as a standalone resource).
- A **Computer Vision** resource.

To use your deployed resource in an application, you must connect to its **endpoint** using either **key-based authentication** or **Microsoft Entra ID authentication**. You can find the endpoint for your resource in the Azure portal, or if you're working in an Azure AI Foundry project, in the Azure AI Foundry portal. **The endpoint is in the form of a URL**, and typically looks something like this:

`https://<resource_name>.cognitiveservices.azure.com/`

After establishing a connection, you can use the OCR feature by calling the **ImageAnalysis** function (via the REST API or with an equivalent SDK method), passing the image URL or binary data, and optionally specifying the language the text is written in (with a default value of en for English).

`https://<endpoint>/computervision/imageanalysis:analyze?features=read&...`

To use the Azure AI Vision Python SDK to extract text from an image, install the **azure-ai-vision-imageanalysis** package. Then, in your code, use either **key-based authentication** or **Microsoft Entra ID authentication** to connect an **ImageAnalysisClient** object to an Azure AI Vision resource. To find and read text in an image, call the **analyze** (or **analyze_from_url**) method, specifying the **VisualFeatures.READ** enumeration.

```
from azure.ai.vision.imageanalysis import ImageAnalysisClient
from azure.ai.vision.imageanalysis.models import VisualFeatures
from azure.core.credentials import AzureKeyCredential

client = ImageAnalysisClient(endpoint="<YOUR_RESOURCE_ENDPOINT>", credential=AzureKeyCredential("<YOUR_AUTHORIZATION_KEY>"))

result = client.analyze(image_data=<IMAGE_DATA_BYTES>, # Binary data from your image file visual_features=[VisualFeatures.READ], lang=
```

The results of the Read OCR function are returned **synchronously**, either as **JSON** or the **language-specific object** of a similar structure. These results are broken down in **blocks** (with the current service only using one block), then **lines**, and then **words**. Additionally, the text values are included at both the **line** and **word** levels, making it easier to read entire lines of text if you don't need to extract text at the individual word level.

Exercise - Read text in images

Now it's your turn to try using the OCR capabilities of Azure AI Vision.

In this exercise, you use the Azure AI Vision Image Analysis SDK to develop a client application that extracts text from images.

Read text in images

Optical character recognition (OCR) is a subset of computer vision that deals with reading text in images and documents. The Azure AI Vision Image Analysis service provides an API for reading text, which you'll explore in this exercise.

Module assessment

1. Which service should you use to locate and read text in signs within a photograph of a street? **Azure AI Vision Image Analysis**
2. Which visual feature enumeration should you use to return OCR results from an image analysis call? **VisualFeatures.Read**
3. Text location information in an image is returned at which levels by Azure AI Vision Image Analysis? **A block containing the location of lines of text as well as individual words.**

Summary

In this module, you learned how to provision an Azure AI Vision resource and use it from a client application to extract text from images.

To learn more about using Azure AI Vision for OCR, see the [OCR - Optical Character Recognition](#) in the Azure AI Vision documentation.

Detect, analyze, and recognize faces

The ability for applications to **detect human faces, analyze facial features and emotions, and identify individuals** is a key artificial intelligence capability.

Learning objectives

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

- Describe the capabilities of the **Azure AI Vision Face service**.
- Write code to **detect and analyze faces** in an image.
- Describe **facial recognition** support in Azure AI Vision Face.
- Describe **responsible AI** considerations when developing facial solutions.

Introduction

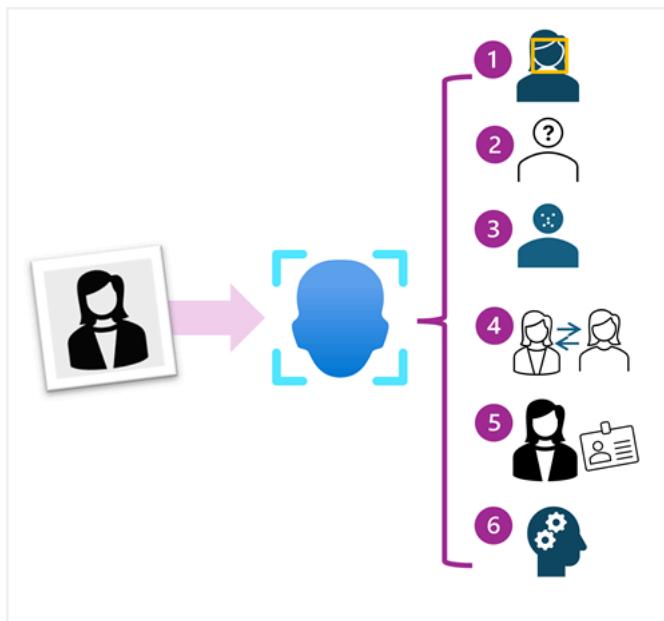
Face detection, analysis, and recognition are all common computer vision challenges for AI systems. The ability to detect when a person is **present**, analyze a person's **facial features**, or **recognize an individual** based on their face is a key way in which AI systems can exhibit human-like behavior and build empathy with users.

In this module, you'll explore how the **Azure AI Vision Face API** enables you to build solutions that analyze faces in images.

Note: Access to the full capabilities of the Face API is restricted in accordance with Microsoft's responsible AI policies. For details, see [Limited Access to Face API](#). This module describes some capabilities that require explicit access. The practical exercise in the module is based on unrestricted features of the service.

Plan a face detection, analysis, or recognition solution

The **Face** service provides comprehensive **facial detection, analysis, and recognition** capabilities.



Face Features

The Face service provides functionality that you can use for:

- Face detection** - for each detected face, the results include an ID that identifies the **face and the bounding box coordinates** indicating its location in the image.
- Face attribute analysis** - you can return a wide range of facial attributes, including:
 - **Head pose** (pitch, roll, and yaw orientation in 3D space)
 - **Glasses** (No glasses, Reading glasses, Sunglasses, or Swimming Goggles)
 - **Mask** (the presence of a face mask)
 - **Blur** (low, medium, or high)
 - **Exposure** (under exposure, good exposure, or over exposure)
 - **Noise** (visual noise in the image)
 - **Occlusion** (objects obscuring the face)
 - **Accessories** (glasses, headwear, mask)
 - **QualityForRecognition** (low, medium, or high)
- Facial landmark location (spatial points)** - coordinates for key landmarks in relation to facial features (for example, **eye corners, pupils, tip of nose**, and so on)
- Face comparison** - you can compare faces across multiple images for similarity (to find individuals with similar facial features) and verification (to determine that a face in one image is the same person as a face in another image)
- Facial recognition** - you can train a model with a collection of faces belonging to specific individuals, and use the model to identify those people in new images.
- Facial liveness** - liveness can be used to determine if the input video is a real stream or a fake to prevent bad-intentioned individuals from spoofing a facial recognition system.

Face detection and recognition models

The Azure AI Vision Face API is built on face detection and recognition models that have been pre-trained. Multiple versions of these models are available, each with specific strengths and capabilities. For example, newer models exhibit greater accuracy when working with small images; but may not provide the same breadth of facial analysis capabilities. When you use the service in an application, you must select the model you want to use based on your requirements.

Tip: For guidance about selecting a detection model, see [Specify a face detection model](#). For guidance about how to select a recognition model, see [Specify a face recognition model](#).

Detect and analyze faces

To use the **Azure AI Vision Face API**, you must provision a resource for the service in an Azure subscription. You can provision Face as a single-service resource, or you can use the Face API in a multi-service Azure AI Services resource; which can be provisioned as a standalone resource or as part of an Azure AI Foundry hub.

To use your resource from a client application you must connect to its **endpoint** using either **key-based authentication** or **Microsoft Entra AI authentication**. When using the REST interface you can provide the authentication key or token in the request header. When using a language-specific SDK (for example, the Python `azure-ai-vision-face` package or the Microsoft .NET `Azure.AI.Vision.Face` package), you use a **FaceClient** object to connect to the service.

```
from azure.ai.vision.face import FaceClient
from azure.ai.vision.face.models import *
from azure.core.credentials import AzureKeyCredential

face_client = FaceClient(endpoint="", credential=AzureKeyCredential("<YOUR_RESOURCE_KEY>"))
```

To detect and analyze faces in an image, you must specify the model-specific features you want the service to return, and then use the client to call the **Detect** method.

```
# Specify facial features to be retrieved
features = [FaceAttributeTypeDetection01.HEAD_POSE, FaceAttributeTypeDetection01.OCCLUSION, FaceAttributeTypeDetection01.AC

# Use client to detect faces in an image
with open("<IMAGE_FILE_PATH>", mode="rb") as image_data: detected_faces = face_client.detect(image_content=image_data.read(),
```

The response from the service depends on:

- The model-specific features requested.
- The number of faces detected in the image.

A response for an image containing a single face might look similar to the following example:

Verify and identify faces

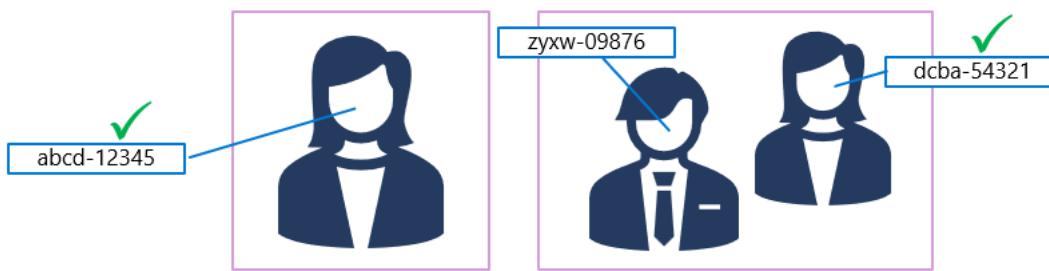
In addition to detecting and analyzing faces, you can use the **Azure AI Vision Face service** to *compare and recognize* faces.

Important: Usage of facial recognition, comparison, and verification requires approval through a Limited Access policy.

Verifying faces

When a face is detected by the Face service, a **unique ID** is assigned to it and **retained in the service resource for 24 hours**. The ID is a GUID, with no indication of the individual's identity other than their facial features.

While the detected face ID is cached, subsequent images can be used to compare the new faces to the cached identity and determine if they're *similar* (in other words, they share similar facial features) or to *verify* that the same person appears in two images.



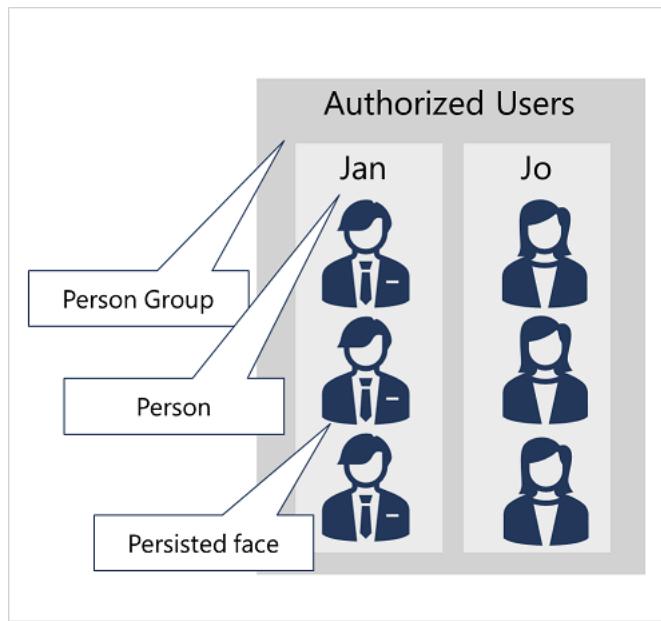
This ability to compare faces anonymously can be useful in systems where it's important to **confirm that the same person is present on two occasions**, without the need to know the actual identity of the person. For example, by taking images of people as they enter and leave a secured space to verify that everyone who entered leaves.

Identifying faces

For scenarios where you need to positively identify individuals, you can train a facial recognition model using face images.

To train a facial recognition model with the Face service:

1. Create a **Person Group** that defines the set of individuals you want to identify (for example, employees).
2. Add a **Person** to the **Person Group** for each individual you want to identify.
3. Add detected faces from multiple images to each **person**, preferably in various poses. The IDs of these faces will no longer expire after 24 hours (so they're now referred to as *persisted* faces).
4. Train the model.



The trained model is stored in your Face (or Azure AI Services) resource, and can be used by client applications to:

1. **Identify** individuals in images.
2. **Verify** the identity of a detected face.
3. Analyze new images to find faces that are **similar** to a known, persisted face.

Tip: To learn more about using face verification and identification to implement a facial recognition solution, see [Face recognition](#) in the Azure AI Vision Face documentation.

Responsible AI considerations for face-based solutions

While all applications of artificial intelligence require considerations for responsible, system that rely on facial or other biometric data can be particularly problematic.

When building a solution that uses facial data, considerations include (but aren't limited to):

- **Data privacy and security.** Facial data is personally identifiable, and should be considered sensitive and private. You should ensure that you have implemented adequate protection for facial data used for model training and inferencing.
- **Transparency.** Ensure that **users are informed** about **how their facial data is used**, and **who will have access** to it.
- **Fairness and inclusiveness.** Ensure that your face-based system can't be used in a manner that is prejudicial to individuals based on their appearance, or to unfairly target individuals.

Note: You can find details of the responsible AI measures taken in the implementation of the Face API in [Data and privacy for Face](#).

Exercise - Detect and analyze faces

Now it's your turn to try using the Azure AI Vision Face service.

In this exercise, you use the Azure AI Vision Face API to develop a client application that detects and analyzes faces in an image.

Detect and analyze faces

The ability to detect and analyze human faces is a core AI capability. In this exercise, you'll explore two **Azure AI Services** that you can use to work with faces in images: the Azure AI Vision service, and the **Face** service.

Module assessment

1. You need to create a facial recognition solution to identify named employees. Which service should you use? **Azure AI Vision Face**
2. What should you return when analyzing an image to see the spatial points related to facial features? **Landmarks**

Summary

In this module, you have learned how to **detect, analyze, and recognize faces**. Facial analysis and recognition solutions can help you address business requirements for security and authorization, digital asset management, and other scenarios. However, you should be mindful of the need to use biometric data such as facial imaging responsibly.

Tip: To find out more about the Azure AI Vision Face service, see the [Face documentation](#).

Classify images

Image classification is used to **determine the main subject of an image**. You can use the Azure **AI Custom Vision services** to train a model that classifies images based on your own categorizations.

Learning objectives

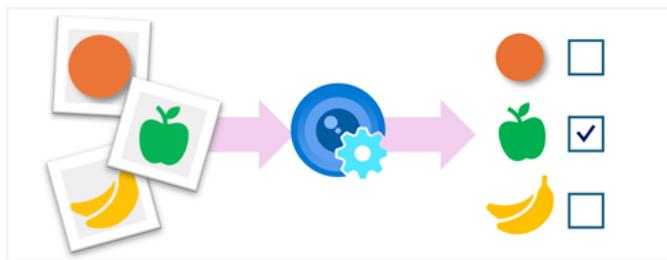
After completing this module, you'll be able to:

- Provision Azure resources for **Azure AI Custom Vision**.
- Train an **image classification model**.
- Use the Azure AI Custom Vision SDK to create an image classification client application.

Introduction

Image classification is a common computer vision problem that requires software to **analyze an image and categorize (or classify) it**.

For example, an unattended checkout system in a grocery store might use a camera to scan each item a customer adds to their cart, and use image classification to identify the product in the image.



In this module, you'll learn how the **Azure AI Custom Vision service** enables you to build your own computer vision models for image classification.

Azure AI Custom Vision

The Azure AI Custom Vision service enables you to build your own computer vision models for **image classification** or **object detection**.

To use the Custom Vision service to create a solution, you need **two Custom Vision resources** in your Azure subscription:

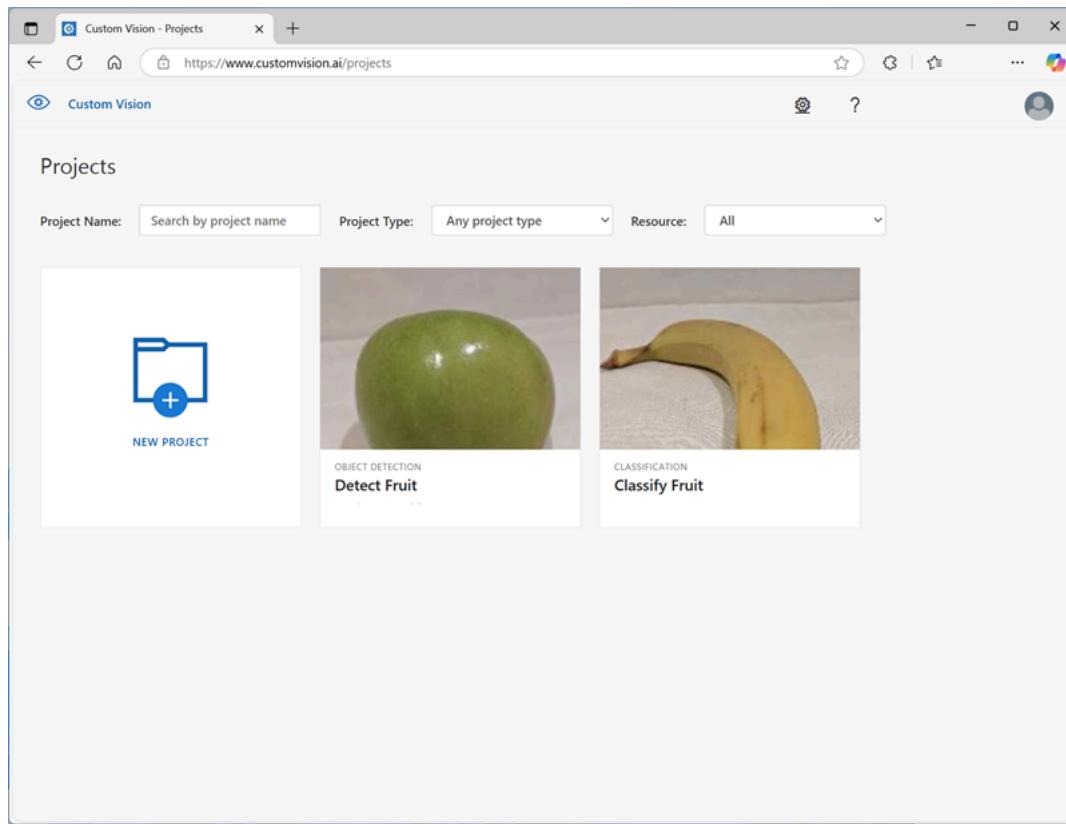
- An **Azure AI Custom Vision training** resource - used to **train a custom model** based on your own training images.
- An **Azure AI Custom Vision prediction** resource - used to **generate predictions** from new images based on your trained model.

When you provision the Azure AI Custom Vision service in an Azure subscription, you can choose to create one or both of these resources. This separation of training and prediction provides flexibility. For example, you can use a training resource in one region to train your model using your own image data; and then deploy one or more prediction resources in other regions to support computer vision applications that need to use your model.

Each resource has its own **unique endpoint and authentication keys**; which are used by client applications to connect and authenticate to the service.

The Custom Vision portal

Azure AI Custom Vision provides a web-based portal, in which you can **train, publish, and test custom vision models**.



You can sign into the [Custom Vision portal](#) using your Azure credentials and use it to create image classification or object detection projects that use Azure AI Custom Vision resources in your Azure subscription.

Each project has a unique project ID; which is used by client applications to perform training or prediction tasks using code.

Custom Vision SDKs

You can write code to train and consume custom models by using the Azure AI Custom Vision language-specific SDKs.

For example, Microsoft C# developers can use the [Microsoft.Azure.CognitiveServices.Vision.CustomVision.Training](#) and [Microsoft.Azure.CognitiveServices.Vision.CustomVision.Prediction](#) Microsoft .NET packages for training and prediction respectively.

Python developers can perform both training and prediction tasks by using the [azure-cognitiveservices-vision-customvision](#) package.

Train an image classification model

Image classification is a computer vision technique in which a model is trained to predict a class label for an image based on its contents. Usually, the class label relates to the main subject of the image.

For example, the following images have been classified based on the type of fruit they contain.



Apple

Banana

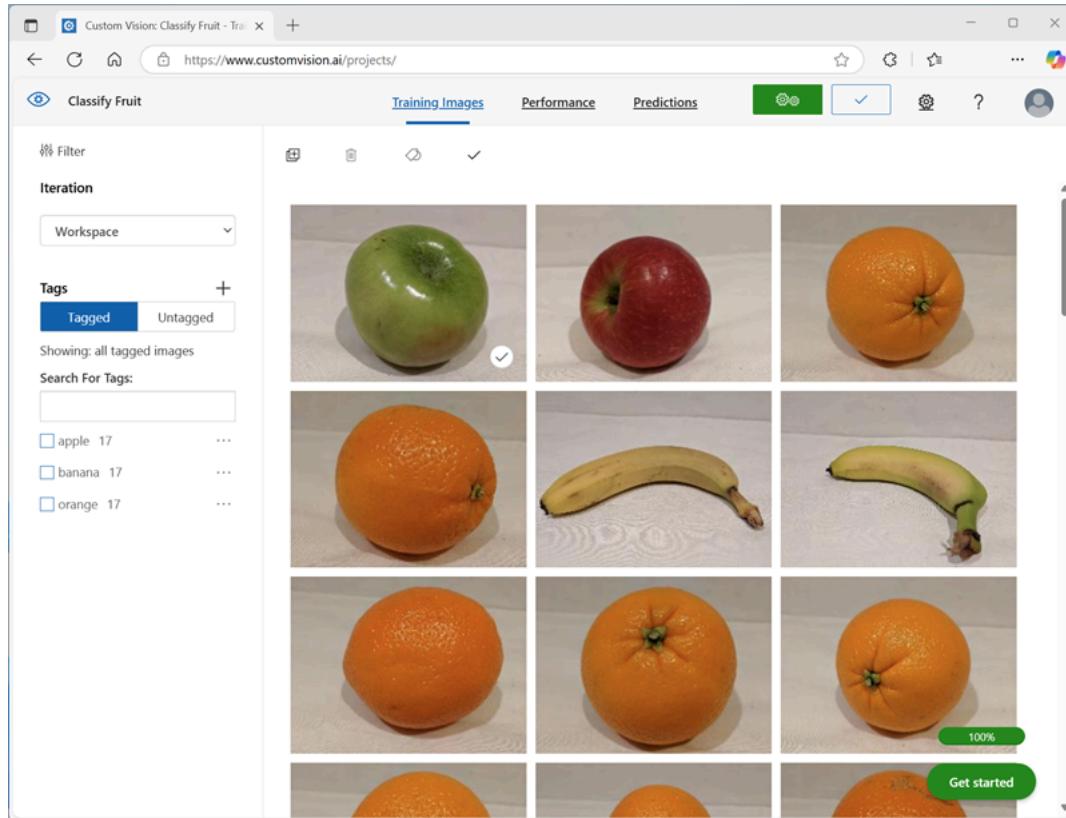
Orange

Models can be trained for **multiclass classification** (in other words, there are multiple classes, but each image can belong to only one class) or **multilabel classification** (in other words, an image might be associated with multiple labels).

Training an image classification model

To train an image classification model with the **Azure AI Custom Vision** service, you can use the **Azure AI Custom Vision portal**, the **Azure AI Custom Vision REST API or SDK**, or a combination of both approaches.

In most cases, you'll typically use the Azure AI Custom Vision portal to train your model.



The portal provides a graphical interface that you can use to:

1. **Create** an image classification project for your model and associate it with a training resource.
2. **Upload** images, **assigning** class label tags to them.
3. **Review** and edit tagged images.
4. **Train and evaluate** a classification model.
5. **Test** a trained model.
6. **Publish** a trained model to a prediction resource.

The REST API and SDKs enable you to perform the same tasks by writing code, which is useful if you need to automate model training and publishing as part of a DevOps process.

Create an image classification client application

After you've trained an image classification model, you can use the Azure AI Custom Vision SDK to develop a client application that submits new images to be classified.

```
from msrest.authentication import ApiKeyCredentials
from azure.cognitiveservices.vision.customvision.prediction import CustomVisionPredictionClient

# Authenticate a client for the prediction API
credentials = ApiKeyCredentials(in_headers={"Prediction-key": "<YOUR_PREDICTION_RESOURCE_KEY>"})
prediction_client = CustomVisionPredictionClient(endpoint="<YOUR_PREDICTION_RESOURCE_ENDPOINT>", credentials=credentials)
```

```

# Get classification predictions for an image
image_data = open("<PATH_TO_IMAGE_FILE>", "rb").read()
results = prediction_client.classify_image("<YOUR_PROJECT_ID>",
                                         "<YOUR_PUBLISHED_MODEL_NAME>",

# Process predictions
for prediction in results.predictions:
    if prediction.probability > 0.5:
        print(image, ': {} {:.0%}'.format(prediction.tag_name, prediction.probability))

```

Exercise - Classify images

Now it's your turn to try using the Azure AI Custom Vision service.

In this exercise, you train and publish a custom image classification model, and use the Azure AI Custom Vision SDK to test it.

Classify images

The **Azure AI Custom Vision** service enables you to create computer vision models that are trained on your own images. You can use it to train image classification and object detection models; which you can then publish and consume from applications.

In this exercise, you will use the *Custom Vision* service to train an image classification model that can identify three classes of fruit (apple, banana, and orange).

In the code file, update the configuration values it contains to reflect the **Endpoint** and an **authentication Key** for your Custom Vision training resource, and the **Project ID** for the custom vision project you created previously.

```

PredictionEndpoint=YOUR_PREDICTION_ENDPOINT
PredictionKey=YOUR_PREDICTION_KEY
ProjectID=YOUR_PROJECT_ID
ModelName=fruit-classifier

```

Module assessment

1. What Azure AI Custom Vision resources should you create in Azure to create a custom vision solution? **A Custom Vision resource for training, and another for prediction.**
2. You want to train a model that can categorize an image as "cat" or "dog" based on its subject. What kind of Azure AI Custom Vision project should you create? **Image classification (multiclass)**
3. What information does a client application need to connect to Azure AI Custom Vision and classify an image? **The endpoint and key for the Custom Vision prediction resource, the project ID for your image classification project, and the name of your deployed model.**

Summary

In this module, you learned how to use the Azure AI Custom Vision service to build your own custom vision models for image classification.

Detect objects in images

Object detection is used to **locate and identify objects in images**. You can use **Azure AI Custom Vision** to train a model to detect specific classes of object in images.

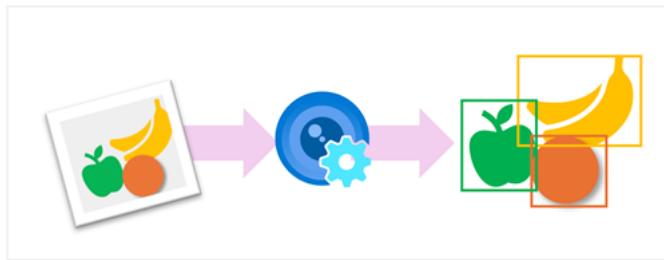
Learning objectives

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

- Provision Azure resources for **Azure AI Custom Vision**
- Understand **object detection**
- Train an **object detector**
- Use the Azure AI Custom Vision SDK to create an **object detection client application**

Introduction

Object detection is a common computer vision problem that requires software to **identify the location of specific classes of object in an image**.



For example, an automated checkout system in a grocery store might use a camera to monitor a checkout conveyor belt on which there might be multiple different items at any one time. The system could use object detection to identify which items are on the belt, and where in the image they appear.

In this module, you'll learn how to use the Azure AI Custom Vision service to create object detection models.

Use Azure AI Custom Vision for object detection

To use the Custom Vision service to create an object detection solution, you need two Custom Vision resources in your Azure subscription:

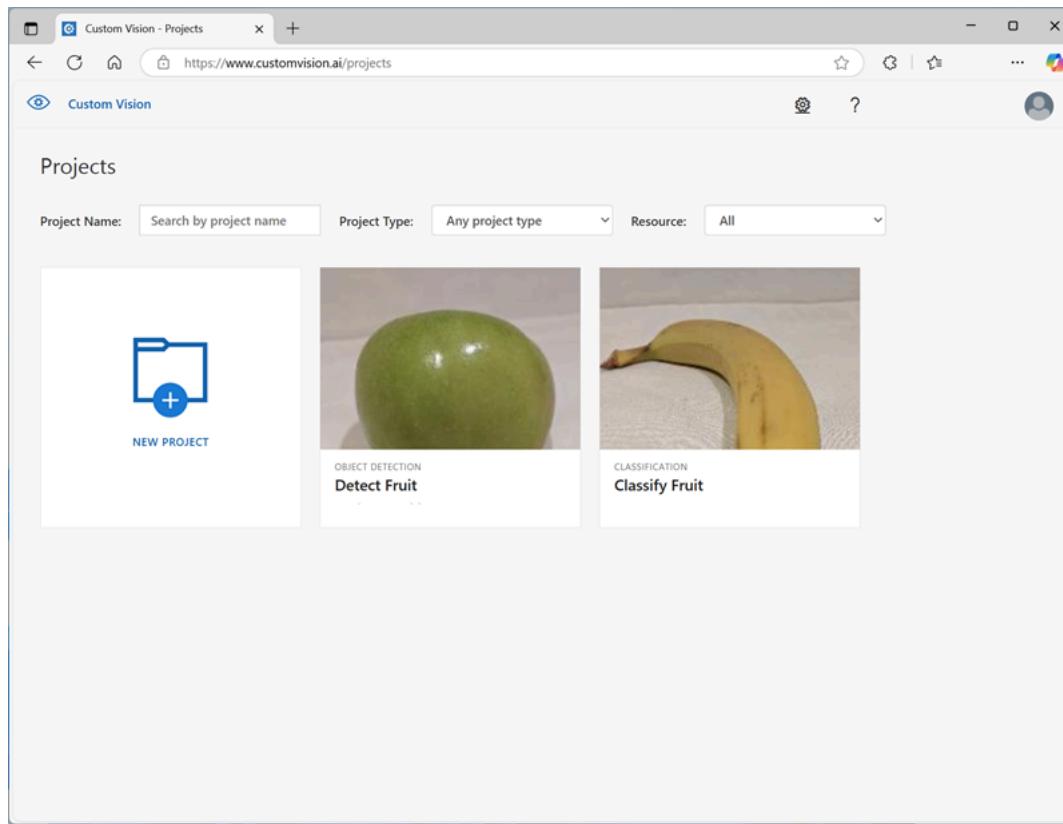
- An **Azure AI Custom Vision training** resource - used to train a custom model based on your own training images.
- An **Azure AI Custom Vision prediction** resource - used to generate predictions from new images based on your trained model.

When you provision the Azure AI Custom Vision service in an Azure subscription, you can choose to create one or both of these resources. This separation of training and prediction provides flexibility. For example, you can use a training resource in one region to train your model using your own image data; and then deploy one or more prediction resources in other regions to support computer vision applications that need to use your model.

Each resource has its own unique **endpoint and authentication keys**; which are used by client applications to connect and authenticate to the service.

The Custom Vision portal

Azure AI Custom Vision provides a **web-based portal**, in which you can train, publish, and test custom vision models.



You can sign into the [Custom Vision portal](#) using your Azure credentials and use it to create image classification or object detection projects that use Azure AI Custom Vision resources in your Azure subscription.

Each project has a unique **project ID**; which is *used by client applications to perform training or prediction tasks using code*.

Custom Vision SDKs

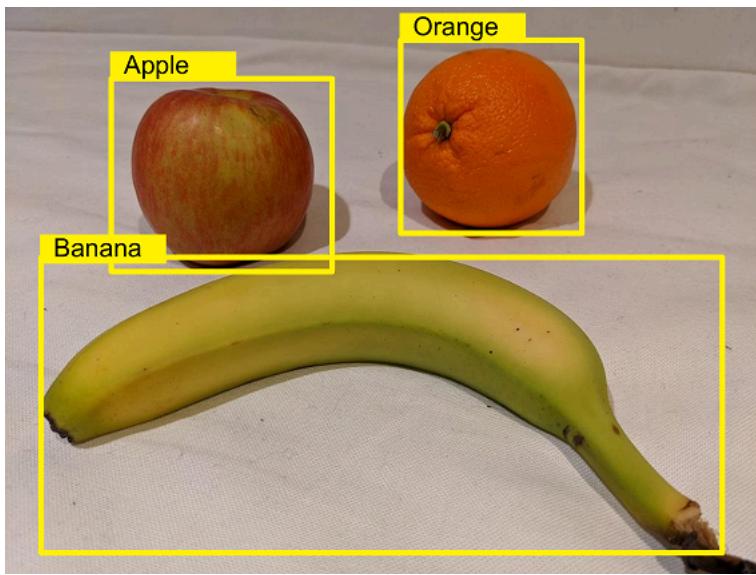
You can write code to train and consume custom models by using the Azure AI Custom Vision language-specific SDKs.

For example, Microsoft C# developers can use the [Microsoft.Azure.CognitiveServices.Vision.CustomVision.Training](#) and [Microsoft.Azure.CognitiveServices.Vision.CustomVision.Prediction](#) Microsoft .NET packages for training and prediction respectively.

Python developers can perform both training and prediction tasks by using the [azure-cognitiveservices-vision-customvision](#) package.

Train an object detector

Object detection is a form of computer vision in which a model is trained to **detect the presence and location of one or more classes of object in an image**.



There are two components to an object detection prediction:

- The **class label** of each object detected in the image. For example, you might ascertain that an image contains an apple, an orange, and a banana.
- The **location** of each object within the image, indicated as coordinates of a bounding box that encloses the object.

To train an object detection model, you can use the [Azure AI Custom Vision portal](#) to upload and label images before training, evaluating, testing, and publishing the model; or you can use the [REST API](#) or a language-specific [SDK](#) to write code that performs the training tasks.

Image labeling

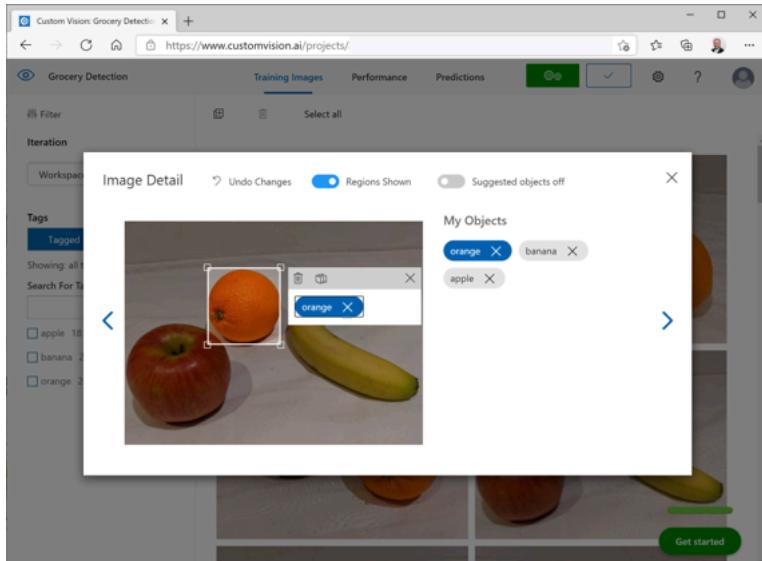
You can use Azure AI Custom Vision to create projects for *image classification* or *object detection*.

The most significant **difference between training an image classification model and training an object detection model** is the **labeling of the images with tags**. While **image classification** requires **one or more tags** that apply to the whole image, **object detection** requires that each label consists of **a tag and a region that defines the bounding box** for each object in an image.

Functions	Requirements
Image classification	One or more tags
Object detection	A tag and a region that defines the bounding box.

Labeling images in the Azure AI Custom Vision portal

The Azure AI Custom Vision portal provides a graphical interface that you can use to label your training images.



The easiest option for labeling images for object detection is to use the **interactive interface in the Azure AI Custom Vision portal**. This interface automatically suggests regions that contain objects, to which you can assign tags or adjust by dragging the bounding box to enclose the object you want to label.

Additionally, after tagging an initial batch of images, you can train the model. Subsequent labeling of new images can benefit from the **smart labeler tool** in the portal, which can suggest not only the regions, but the classes of object they contain.

Alternative labeling approaches

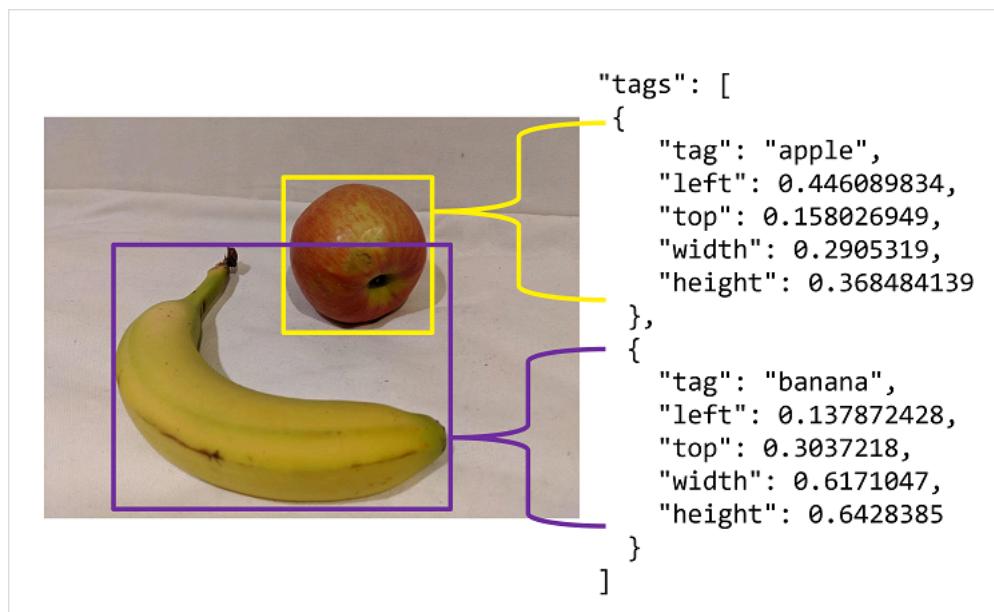
Alternatively, you can use **a custom or third-party labeling tool**, or choose to **label images manually**, to take advantage of other features, such as assigning image labeling tasks to multiple team members.

If you choose to use a labeling tool other than the Azure AI Custom Vision portal, you may need to adjust the output to match the measurement units expected by the Azure AI Custom Vision API. **Bounding boxes** are defined by **four values that represent the left (X) and top (Y) coordinates of the top-left corner of the bounding box**, and **the width and height of the bounding box**. These values are expressed as **proportional values relative to the source image size**. For example, consider this bounding box:

- Left: 0.1
- Top: 0.5
- Width: 0.5
- Height: 0.25

This defines a box in which the left is located 0.1 (one tenth) from the left edge of the image, and the top is 0.5 (half the image height) from the top. The box is half the width and a quarter of the height of the overall image.

The following image shows labeling information in JSON format for objects in an image.



Develop an object detection client application

After you've trained an object detection model, you can use the **Azure AI Custom Vision SDK to develop a client application** that submits new images to be analyzed.

```
from msrest.authentication import ApiKeyCredentials  
from azure.cognitiveservices.vision.customvision.prediction import CustomVisionPredictionClient  
  
# Authenticate a client for the prediction API  
credentials = ApiKeyCredentials(in_headers={"Prediction-key": "<YOUR_PREDICTION_RESOURCE_KEY>"}) prediction_client = CustomVisionPredictionClient(e  
  
# Get classification predictions for an image  
image_data = open("<PATH_TO_IMAGE_FILE>"), "rb").read() results = prediction_client.detect_image("<YOUR_PROJECT_ID>",


```

```
# Process predictions
for prediction in results.predictions:
    if prediction.probability > 0.5:
        left = prediction.bounding_box.left
        top = prediction.bounding_box.top
        height = prediction.bounding_box.height
        width = prediction.bounding_box.width
        print(f"{prediction.tag_name} ({prediction.probability})")
        print(f"  Left:{left}, Top:{top}, Height:{height}, Width:{width}")
```

Exercise - Detect objects in images

If you have an Azure subscription, you can use Azure AI Custom Vision to create a custom object detection model for yourself.

In this exercise, you'll train an object detection model and test it from a client application.

Detect objects in images

The **Azure AI Custom Vision** service enables you to create computer vision models that are trained on your own images. You can use it to train image classification and object detection models; which you can then publish and consume from applications.

In this exercise, you will use the Custom Vision service to train an object detection model that can detect and locate three classes of fruit (apple, banana, and orange) in an image.

Module assessment

1. What does an object detection model predict? **The location and class of specific classes of object in an image.**
2. What must you do before taking advantage of the smart labeler tool when creating an object detection model? **Tag some images with objects of each class and train an initial object detection model.**

Summary

In this module, you have learned how to use the Azure AI Custom Vision service to create object detection models.

Tip: To find out more about the Azure AI Custom Vision service, see the [Azure AI Custom Vision documentation](#).

Analyze video

Azure Video Indexer is a service to extract insights from video, including **face identification, text recognition, object labels, scene segmentations**, and more.

Learning objectives

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

- Describe Azure Video Indexer capabilities
- Extract custom insights
- Use Azure Video Indexer widgets and APIs

Introduction

It's increasingly common for organizations and individuals to generate content in video format. For example, you might use a cellphone to capture a live event, or you might record a teleconference that combines webcam footage and presentation of slides or documents. As a result, a great deal of information is encapsulated in video files, and you may need to extract this information for analysis or to support indexing for searchability.

In this module, you will learn how to use the **Azure Video Indexer service** to analyze videos.

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

- Describe Azure Video Indexer capabilities.
- Extract custom insights.
- Use Azure Video Indexer widgets and APIs.

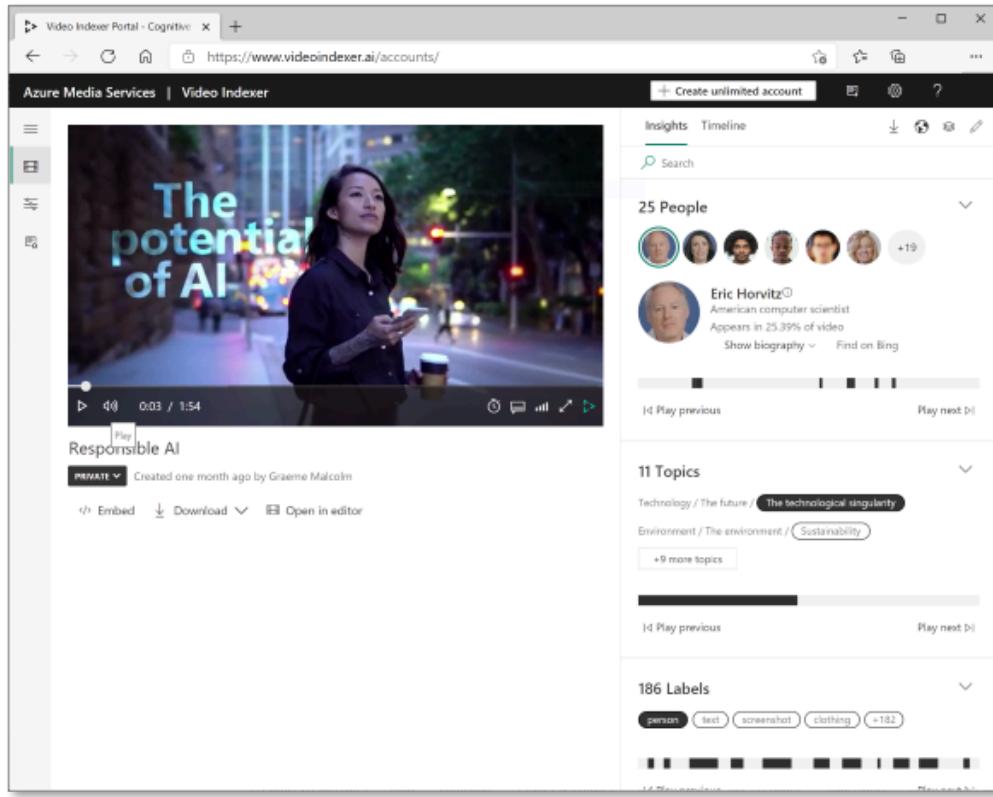
Understand Azure Video Indexer capabilities

The **Azure Video Indexer service** is designed to help you extract information from videos. It provides functionality that you can use for:

- **Facial recognition** - detecting the presence of individual people in the image. This requires Limited Access approval.
- **Optical character recognition** - reading text in the video.
- **Speech transcription** - creating a text transcript of spoken dialog in the video.
- **Topics** - identification of key topics discussed in the video.
- **Sentiment** - analysis of how positive or negative segments within the video are.
- **Labels** - label tags that identify key objects or themes throughout the video.

- **Content moderation** - detection of adult or violent themes in the video.
- **Scene segmentation** - a breakdown of the video into its constituent scenes.

The Video Analyzer service provides a portal website that you can use to upload, view, and analyze videos interactively.



Extract custom insights

Azure Video Indexer includes predefined models that can **recognize well-known celebrities, do OCR, and transcribe spoken phrases into text**. You can extend the recognition capabilities of Video Analyzer by creating custom models for:

- **People**. Add images of the faces of people you want to recognize in videos, and train a model. Video Indexer will then recognize these people in all of your videos.

Note: This only works after Limited Access approval, adhering to our Responsible AI standard.

- **Language**. If your organization uses specific terminology that may not be in common usage, you can train a custom model to detect and transcribe it.
- **Brands**. You can train a model to recognize specific names as brands, for example to identify products, projects, or companies that are relevant to your business.

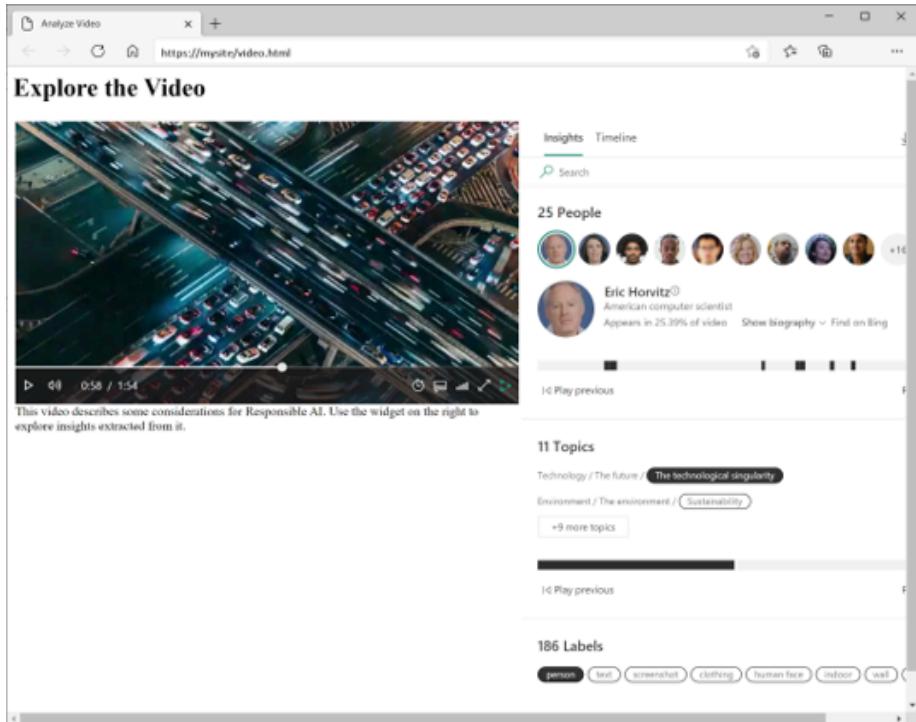
Note: People only works after [Limited Access](#) approval, adhering to our Responsible AI standard.

Use Video Analyzer widgets and APIs

While you can perform all video analysis tasks in the Azure Video Indexer portal, you may want to incorporate the service into custom applications. There are two ways you can accomplish this.

Azure Video Indexer widgets

The **widgets** used in the [Azure Video Indexer portal](#) to play, analyze, and edit videos can be embedded in your own custom HTML interfaces. You can use this technique to share insights from specific videos with others without giving them full access to your account in the Azure Video Indexer portal.



Azure Video Indexer API

Azure Video Indexer provides a [REST API](#) that you can use to obtain information about your account, including an [access token](#).

<https://api.videoindexer.ai/Auth/<location>/Accounts/<accountId>/AccessToken>

You can then use your token to consume the REST API and automate video indexing tasks, creating projects, retrieving insights, and creating or deleting custom models.

For example, a GET call to

<https://api.videoindexer.ai/<location>/Accounts/<accountId>/Customization/CustomLogos/Logos/<logoId>?<accessToken>>

REST endpoint returns the specified logo.

In another example, you can send a GET request to

<https://api.videoindexer.ai/<location>/Accounts/<accountId>/Videos?<accessToken>>

which returns details of videos in your account, similar to the following JSON example:

Deploy with ARM template

Azure Resource Manager (ARM) templates are available to create the Azure AI Video Indexer resource in your subscription, based on the parameters specified in the template file.

For a full list of available APIs, see the [Video Indexer Developer Portal](#).

Exercise - Analyze video

Now it's your turn to try using the Azure AI Video Indexer service.

In this exercise, you analyze a video using the Azure AI Video Indexer portal and its API.

Analyze video

A large proportion of the data created and consumed today is in the format of video. **Azure AI Video Indexer** is an AI-powered service that you can use to index videos and extract insights from them.

Module assessment

1. You want Azure Video Indexer to analyze a video. What must you do first? **Upload the video to Azure Video Indexer and index it.**
2. You want Azure Video Indexer to recognize brands in videos recorded from conference calls. What should you do? **Edit the Brands model to show brands suggested by Bing, and add any new brands you want to detect.**

Summary

In this module, you learned how to use the **Azure Video Indexer** service to analyze videos.

Now that you've completed this module, you can:

- Describe Azure Video Indexer capabilities.
- Extract custom insights.
- Use **Azure Video Indexer widgets** and **APIs**.

To find out more about the Azure Video Indexer service, see the [Azure Video Indexer documentation](#).

Develop a vision-enabled generative AI application

A picture says a thousand words, and **multimodal generative AI models** can interpret images to respond to visual prompts. Learn how to build **vision-enabled chat apps**.

Learning objectives

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

- Deploy a **vision-enabled generative AI model** in Azure AI Foundry.
- Test an **image-based prompt** in the chat playground.
- Create a chat app that submits **image-based prompts**.

Introduction

Generative AI models enable you to develop chat-based applications that reason over and respond to input. Often this input takes the form of a text-based prompt, but increasingly **multimodal models that can respond to visual input** are becoming available.

In this module, we'll discuss **vision-enabled generative AI** and explore how you can use Azure AI Foundry to create generative AI solutions that respond to prompts that include **a mix of text and image data**.

Deploy a multimodal model

To handle prompts that include images, you need to deploy a **multimodal generative AI model** - in other words, a model that supports **not only text-based input, but image-based (and in some cases, audio-based) input** as well. Multimodal models available in Azure AI Foundry include (among others):

- Microsoft Phi-4-multimodal-instruct

- OpenAI gpt-4o
- OpenAI gpt-4o-mini

Tip: To learn more about available models in Azure AI Foundry, see the [Model catalog and collections in Azure AI Foundry portal](#) article in the Azure AI Foundry documentation.

Testing multimodal models with image-based prompts

After deploying a multimodal model, you can test it in the chat playground in Azure AI Foundry portal.

The screenshot shows the Azure AI Foundry Chat playground interface. On the left, there's a sidebar with navigation links like Overview, Model catalog, Playgrounds (which is selected), AI Services, etc. The main area has tabs for View code, Evaluate, Deploy, Import, Export, and Build with templates. A deployment dropdown shows "gpt-4o (version:2024-11-20)". The Chat history section shows a large image of a mango. Below the image, a text message asks, "What desserts could I make with this fruit?". A response box below the mango image says, "The fruit in the image is a mango! Mangoes are incredibly versatile and can be used in a variety of desserts. Here are some ideas: 1. **Mango Sorbet** Blend mango puree with sugar and a splash of lime juice, then freeze it into a refreshing sorbet. 2. **Mango Cheesecake** Use mango puree as a topping or swirl it into the batter for a tropical twist on classic cheesecake. 3. **Mango Sticky Rice** Combine coconut milk, sugar, and sticky rice, then top with fresh mango slices". At the bottom, there's another image of a dragon fruit and a text input field asking, "What about this one?". A status bar at the bottom right indicates "644/128000 tokens to be sent".

In the chat playground, you can upload an image from a local file and add text to the message to elicit a response from a multimodal model.

Develop a vision-based chat app

To develop a client app that engages in **vision-based chats with a multimodal model**, you can use the same basic techniques used for text-based chats. You require a connection to the endpoint where the model is deployed, and you use that **endpoint** to submit prompts that consists of messages to the model and process the responses.

The **key difference** is that prompts for a *vision-based chat* include *multi-part user messages that contain both a text (or audio where supported) content item and an image content item*.

The JSON representation of a prompt that includes a multi-part user message looks something like this:

```
{
  "messages": [
    { "role": "system", "content": "You are a helpful assistant." },
    { "role": "user", "content": [
      {
        "type": "text",
        "text": "Describe this picture:"
      },
      {
        "type": "image_url",
        "image_url": {
          "url": "https://....."
        }
      }
    ] }
  ]
}
```

The image content item can be:

- A URL to an image file in a web site.
- Binary image data

When using binary data to submit a local image file, the **image_url content** takes the form of a **base64 encoded value** in a data URL format:

```
{  
  "type": "image_url",  
  "image_url": {  
    "url": "data:image/jpeg;base64,<binary_image_data>"  
  }  
}
```

Depending on the model type, and where you deployed it, you can use **Microsoft Azure AI Model Inference** or **OpenAI APIs** to submit **vision-based prompts**. These libraries also provide language-specific SDKs that abstract the underlying REST APIs.

In the exercise that follows in this module, you can use the Python or .NET SDK for the **Azure AI Model Inference API** and the **OpenAI API** to develop a vision-enabled chat application.

Exercise - Develop a vision-enabled chat app

Develop a vision-enabled chat app

In this exercise, you use the **Phi-4-multimodal-instruct** generative AI model to generate responses to prompts that include images. You'll develop an app that provides AI assistance with fresh produce in a grocery store by using Azure AI Foundry and the Azure AI Model Inference service.

Module assessment

1. Which kind of model can you use to respond to visual input? **Multimodal models**.
2. How can you submit a prompt that asks a model to analyze an image? **Submit a prompt that contains a multi-part user message, containing both text content and image content.**
3. How can you include an image in a message? **As a URL or as binary data.**

Summary

In this module, you learned about **vision-enabled generative AI models** and how to implement chat solutions that include image-based input.

Vision-enabled models let you create AI solutions that can **understand images and respond to related questions or instructions**. Beyond just identifying objects in pictures, some models can also use reasoning based on what they see. For instance, they can interpret a chart or assess if an object is damaged.

Tip: For more information about working with multimodal models in Azure AI Foundry, see [How to use image and audio in chat completions with Azure AI model inference](#) and [Quickstart: Use images in your AI chats](#).

Generate images with AI

In **Azure AI Foundry**, you can use image generation models to create original images based on natural language prompts.

Learning objectives

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

- Describe the capabilities of image generation models
- Use the Images playground in Azure AI Foundry portal
- Integrate image generation models into your apps

Introduction

With Azure AI Foundry, you can use language models to generate content based on natural language prompts. Often the generated content is in the form of natural language text, but increasingly, models can generate other kinds of content.

For example, the OpenAI DALL-E image generation model can create original graphical content based on a description of a desired image.

The ability to **use AI to generate graphics has many applications**; including

- the creation of illustrations or photorealistic images for articles or marketing collateral
- generation of unique product or company logos, or
- any scenario where a desired image can be described.

In this module, you'll learn how to develop an application that uses generative AI to generate original images.

What are image-generation models?

Azure AI Foundry supports multiple models that are capable of generating images, including (but not limited to):

- DALL-E 3
- GPT-Image 1

Tip: For the latest information about model availability in Azure AI Foundry, view the model catalog. See [Model catalog and collections in Azure AI Foundry portal](#) for details.

Image generation models are generative AI model that can create graphical data from natural language input. Put more simply, you can provide the model with a description and it can generate an appropriate image.

For example, you might submit the following natural language prompt to an image generation model:

```
A squirrel on a motorcycle
```

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

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

Explore image-generation models in Azure AI Foundry portal

To experiment with image generation models, you can create an Azure AI Foundry project and use the **Images playground** in Azure AI Foundry portal to submit prompts and view the resulting generated images.

When using the playground, you can adjust the **settings** to control the output. For example, when using a DALL-E model you can specify:

- The **resolution (size)** of the generated images. Available sizes are **1024x1024 (which is the default value), 1792x1024, or 1024x1792**.

- The **image style** to be generated (such as vivid or natural).
- The **image quality** (choose from standard or HD).

Create a client application that uses an image generation model

You can use a **REST API** to consume DALL-E models from applications. Alternatively, you can use a **language-specific SDK** (for example, the OpenAI Python SDK or the Azure OpenAI .NET SDK) to abstract the REST methods.

You initiate the image generation process by **submitting a request to the service endpoint with the authorization key in the header. The request contains parameters describing the image-generation requirements**. For example, parameters for a DALL-E model include:

- **prompt**: The description of the image to be generated.
- **n**: The number of images to be generated. DALL-E 3 only supports n=1.
- **size**: The resolution of the image(s) to be generated (1024x1024, 1792x1024, or 1024x1792 for DALL-E 3)
- **quality** Optional: The quality of the image (standard or hd). Defaults to standard.
- **style** Optional: The visual style of the image (natural or vivid). Defaults to vivid.

For example, the following JSON could be submitted via the REST API to a DALL-E model, prompting it to generate an 1024 x 1024 image of a badger wearing a tuxedo:

```
{ "prompt": "A badger wearing a tuxedo", "n": 1, "size": "1024x1024", "quality": "hd", "style": "vivid" }
```

With DALL-E 3, the result from the request is **processed synchronously** with the response containing the URL for the generated image. The response is similar to the following JSON:

```
{ "created": 1686780744, "data": [ { "url": "<URL of generated image>", "revised_prompt": "<prompt that was" } ] }
```

The **data** element includes the url value, which references a **PNG image file** generated from the prompt that you can then view or download. The response also contains a revised prompt that was used to generate the image, which was updated by the system to achieve the most desirable results. In this example, the image might look similar to the following image.

Exercise - Generate images with AI

Now it's your chance to use generative AI to create images. In this exercise, you'll provision an Azure AI Foundry project and deploy a **DALL-E model**. Then, you'll explore image generation in the Azure AI Foundry portal. Finally, you'll use the Python or .NET SDK to consume the DALL-E model from a custom application.

In this exercise, you use the the OpenAI DALL-E generative AI model to generate images. You'll develop your app by using Azure AI Foundry and the Azure OpenAI service.

Generate images with AI

In this exercise, you use the **OpenAI DALL-E** generative AI model to generate images. You also use the **OpenAI Python SDK** to create a simple app to generate images based on your prompts.

Module assessment

1. You want to use a model in Azure AI Foundry to generate images. Which model should you use? **DALL-E**
2. Which playground in Azure AI Foundry portal should you use to test an image-generation model? **Images**
3. In a REST request to generate images, what does the n parameter indicate? **The number of images to be generated.**

Summary

This module described **image generation models**, and how you can use them in Azure AI Foundry to generate images based on natural language prompts. You can explore image generation models using the **Images playground** in Azure AI Foundry portal, and you can use **REST APIs or SDKs** to build applications that generate new images.

Tip: To learn more about using DALL-E in the Azure OpenAI service, see [Quickstart: Generate images with Azure OpenAI Service](#) in the Azure OpenAI service documentation.

5.1 Create a multimodal analysis solution with Azure AI Content Understanding

Use **Azure AI Content Understanding** for **multimodal content analysis and information extraction**.

Learning objectives

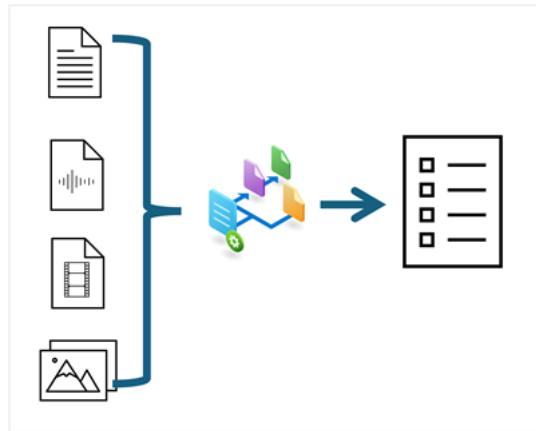
After completing this module, you will be able to:

- Describe capabilities of Azure AI Content Understanding.
- Use Azure AI Content Understanding to build a **content analyzer**.
- Consume a **Content Understanding analyzer** by using the REST API.

Introduction

Organizations today rely on information that is often locked up in content assets such as documents, images, videos, and audio recordings. Extracting information from this content can be challenging, laborious, and time-consuming, and organizations often need to build solutions based on multiple technologies for content analysis depending on the formats being used.

Azure AI Content Understanding is a **multimodal service** that **simplifies the creation of AI-powered analyzers** that can extract information from **content** in practically any format.



In this module, you'll explore the capabilities of Azure AI Content Understanding, and learn how to use it to build custom analyzers.

Note: Azure AI Content Understanding is currently in public preview. Details described in this module are subject to change.

What is Azure AI Content Understanding?

Azure AI Content Understanding is a *generative AI service* that you can use to extract insights and data from multiple kinds of content. With Content Understanding, you can quickly build applications that analyze complex data and generate outputs that can be used to automate and optimize processes.

Content Understanding is a component of Azure AI services. To use it, you need to provision an Azure AI services resource in your Azure subscription. You can develop and manage a Content Understanding solution:

- In the **Azure AI Foundry portal**
- By using the **Content Understanding REST API**

Multimodal content analysis

Content Understanding can extract information from common kinds of content, enabling you to use a single service with a straightforward and consistent development process to **build multimodal content analysis solutions**.

Documents and forms

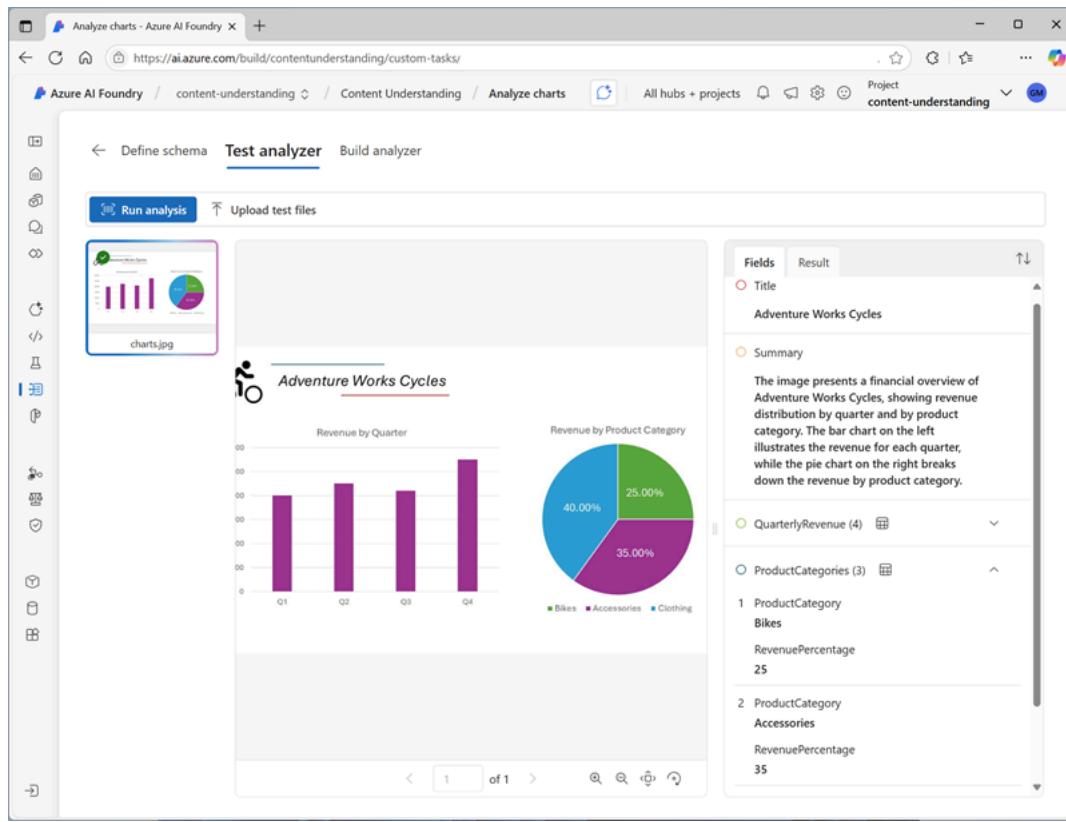
You can use Content Understanding to **analyze documents and forms and retrieve specific field values**. For example, you could extract key data values from an invoice to automate payment processing.

The screenshot shows the Azure AI Foundry interface for 'Invoice Analysis - Azure AI Foundry'. The 'Test analyzer' tab is selected. On the left, there are two PDF files: 'invoice-1234.pdf' and 'invoice-1235.pdf', with 'invoice-1235.pdf' highlighted by a blue box. The main area displays the contents of 'invoice-1235.pdf', which includes a header for 'Contoso Ltd' with address '2 Main St, Bigtown, England, EH1 234' and phone 'Tel: 555 123-4567', followed by an invoice table with items like '42mm Widget' and '5mm screws pack', and a summary table with totals. To the right, a 'Fields' table lists extracted fields with their confidence scores:

Fields	Result	↑
CustomerAddress	p.1	86.90%
	321 Pond Lane Waterville England GL1 010	
CustomerName	p.1	96.80%
	Ava Jones	
InvoiceDate	p.1	99.80%
	2025-07-03	
InvoiceId	p.1	97.20%
	1235	
InvoiceTotal	p.1	97.60%
	115.62	
SubTotal	p.1	98.90%
	91.48	
TotalTax	p.1	98.40%
	9.14	
VendorAddress	p.1	97.80%

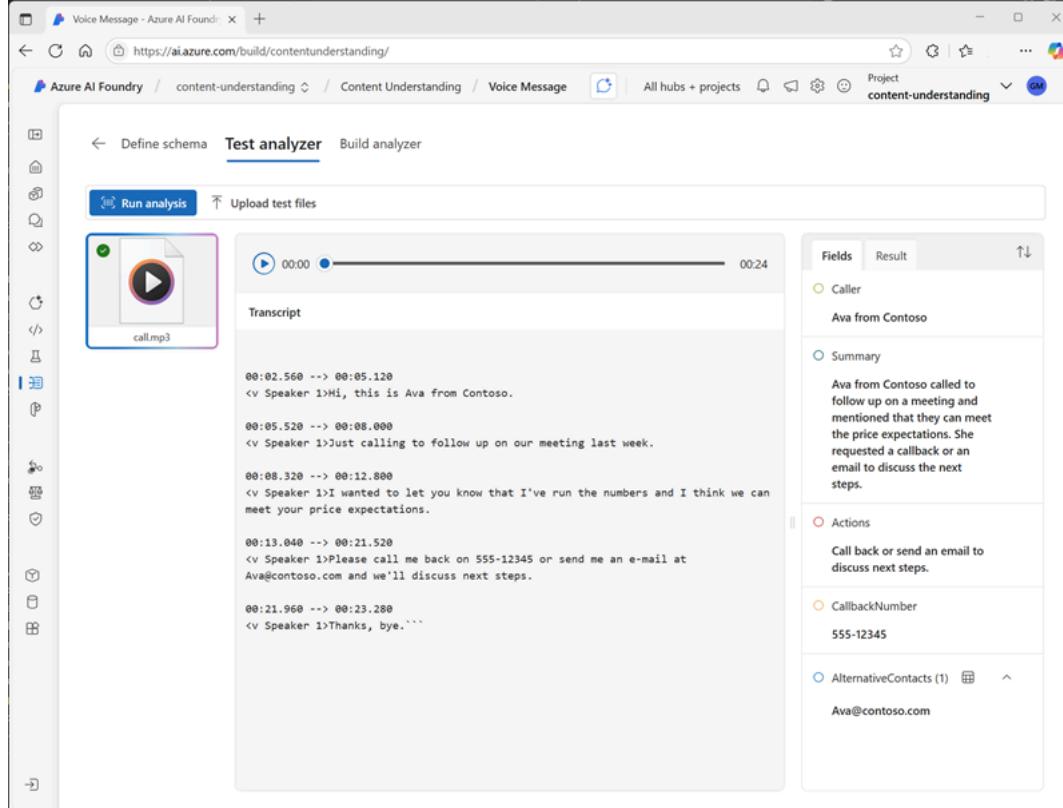
Images

You can analyze images to *infer information from visuals such as charts, identify physical defects in products or other items, detect the presence of specific objects or people, or determine other information visually*.



Audio

Analysis of audio enables you to automate tasks like *summarizing conference calls*, *determining sentiment of recorded customer conversations*, or *extracting key data from telephone messages*.



Video

Video accounts for a large volume of the data captured today, and you can use Content Understanding to **analyze and extract insights from video** to support many scenarios. For example, to *extract key points from video conference recordings*, *to summarize presentations*, or *to detect the presence of*

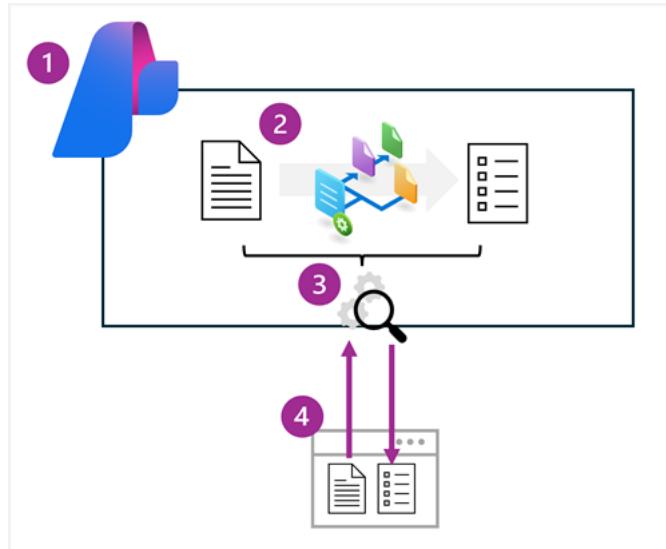
specific activity in security footage.

The screenshot shows the Azure AI Foundry interface with the project 'content-understanding'. On the left, there's a sidebar with various icons. In the center, under 'Test analyzer', there's a video thumbnail for 'meeting-2.mp4' with a play button. Below it is a bar chart titled 'Projected user adoption' showing three bars. A timeline slider is at 00:33. To the right, there's a 'Fields' section and a 'Result' section. The 'Fields' section lists 'Shot 2 00:23' and 'summary'. The 'Result' section contains a narrative about Harry sharing a slide with projected user adoption metrics for the first three months, impressing Lisa. It also lists participants (Harry, Lisa), assigned actions (task: send presentation slide to Lisa), and shared slides (Projected user adoption metrics for the first three months).

Create a Content Understanding analyzer

Content Understanding solutions are based on the creation of an **analyzer**; which is trained to extract specific information from a particular type of content based on a schema that you define.

The high-level process for creating a Content Understanding solution includes the following steps:



1. Create an Azure AI services resource.
2. Define a **Content Understanding schema** for the information to be extracted. This can be based on a **content sample** and a **analyzer template**.
3. Build an **analyzer** based on the completed schema.
4. Use the **analyzer** to extract or generate fields from new content.

Numerous **analyzer templates** are provided to help you develop an appropriate analyzer for your needs quickly. Additionally, because of the generative AI capabilities of Content Understanding, you can use minimal training data to define a schema by example. In many cases, the service accurately identifies the data values in the sample content that map to the schema elements automatically, though you can also explicitly label fields in content such as documents to improve the performance of your analyzer.

Creating an analyzer with Azure AI Foundry

While you can provision an Azure AI services resource and develop a complete Content Understanding solution through the REST API, **the preferred approach for AI development projects is to use Azure AI Foundry**. Specifically, you can **use the Azure AI Foundry portal to create a project, define a Content Understanding schema, and build and test an analyzer**.

1. Creating a Content Understanding project

In Azure AI Foundry, you can *create a project* in an existing AI hub, or you can create a new hub as you create the project. In addition to the AI hub itself, creating a hub provisions the Azure resources needed to support one or more projects; including an Azure AI services resource, storage, and a key vault resource to store sensitive details like credentials and keys.

Note: Content Understanding schemas can only be created in Azure locations where the service is supported. For more information, see [Content Understanding region and language support](#).

2. Defining a schema

After creating a project, **the first step in building an analyzer is to define a schema** for the content the analyzer will process, and the information it will extract. **Azure AI Foundry provides a schema editor interface** in which you can upload a file (document, image, audio, or video) on which the schema should be based. You can then apply an appropriate schema template and define the specific fields you want the analyzer to identify.

Note: The templates and field types available in a schema depend on the content type of the file on which the schema is based. Some content types support additional optional functionality, such as extracting barcodes and formulae from text in documents. For more information about using Content Understanding with different content types, see the following articles in the product documentation:

- [Content Understanding document solutions](#)
- [Content Understanding image solutions](#)
- [Content Understanding audio solutions](#)
- [Content Understanding video solutions](#)

3. Testing

You can **test the analyzer schema** at any time during the development process by running analysis on the sample file used to define the schema or other uploaded files. **The test results include the extracted field values and the JSON format output** returned by the analyzer to client applications.

4. Building an analyzer

When you're satisfied with the performance of your schema, you can build your analyzer. Building an analyzer makes it **accessible to client applications through Content Understanding endpoint** for the Azure AI services resource associated with your project.

After building your analyzer, you can continue to test it in the Azure AI Foundry portal, and refine the schema to create new named versions with different capabilities.

Use the Content Understanding REST API

The **Content Understanding REST API** provides a programmatic interface that you can use to **create, manage, and consume analyzers**.

To use the REST API, your *client application submits HTTP calls to the Content Understanding endpoint* for your Azure AI services resource, **passing one of the authorization keys in the header**. You can obtain the endpoint and keys in the Azure portal or in the Azure AI Foundry portal. You can also use the Azure AI Foundry API to connect to the project and retrieve the endpoint and key for your Azure AI Services resource programmatically.

The screenshot shows the Azure AI Foundry interface for the 'content-understanding' project. On the left, there's a sidebar with various options like Overview, Model catalog, Playgrounds, AI Services, Agents, Templates, Fine-tuning, Content Understanding (selected), Prompt flow, Assess and improve, Tracing, Evaluation, Safety + security, My assets (Models + endpoints, Data + indexes, Web apps), and Management center. The main content area has tabs for 'Endpoints and keys' and 'Project details'. Under 'Endpoints and keys', there's an 'API Key' field containing a redacted string, 'Included capabilities' (Azure AI inference, Azure OpenAI, Azure AI Services selected), and 'Azure AI Services endpoint' set to 'https://xxxxxxxxxxxxxxxxxxxxxx.cognitiveservices.azure.com/'. It also shows 'Speech to text endpoint' at 'https://westus.stt.speech.microsoft.com' and 'Text to speech endpoint' at 'https://westus.tts.speech.microsoft.com'. A link to 'API documentation' is present. Under 'Project details', there's a 'Project connection string' field with a redacted value, a 'Subscription' section showing 'My Subscription', and a 'Management center' button.

Using the REST API to analyze content

One of the most common uses of the REST API is to submit content to an existing analyzer that you have previously built, and retrieve the results of analysis. The analysis request returns an **operation ID** value that represents an asynchronous task. Your client application must then use another request to pass the operation ID back to the endpoint and retrieve the **operation status** - potentially polling multiple times until the operation is complete and the results are returned in JSON format.

For example, to analyze a document, a client application might submit a POST request to the analyze function containing the following JSON body:

```
POST {endpoint}/contentunderstanding/analyzers/{analyzer}:analyze?api-version={api version} { "url": "https://host.com/doc.pdf" }
```

Note: You can specify a URL for the content file location, or you can include the binary contents of the file.

Assuming the request is authenticated and initiated successfully, the response will be similar to this example:

```
Operation-Id: 1234abcd-1234-abcd-1234-abcd1234abcd
```

```
Operation-Location: {endpoint}/contentunderstanding/analyzers/{analyzer}/results/1234abcd-1234-abcd-1234-abcd1234abcd?api-version={api version} { }
```

Your client application must then use the **operation ID** that has been returned to check the status of the operation until it has succeeded (or failed) by submitting a GET request to the results function.

```
GET {endpoint}/contentunderstanding/analyzers/{analyzer}/results/1234abcd-1234-abcd-1234-abcd1234abcd?api-version={api version}
```

When the operation has completed successfully, the response contains a JSON payload representing the results of the analysis. The specific results depend on the content and schema.

Note: For more information about the Content Understanding REST API, see the [reference documentation](#).

Exercise - Extract information from multimodal content

Extract information from multimodal content

In this exercise, you use Azure Content Understanding to extract information from a variety of content types; including an invoice, an images of a slide containing charts, an audio recording of a voice messages, and a video recording of a conference call.

A screenshot of a web browser window titled "Azure AI Foundry" and "Azure-AI-Content-Understanding". The URL is <https://ai.azure.com/?tid=4e1ee3db-4df6-4142-b7b9-bec15f1...>. The page displays a search bar at the top with the placeholder "Search for a model". Below it, a section titled "Latest models" shows six model cards:

- DeepSeek-R1-0528** (Chat completion)
- grok-3** (Chat completion)
- grok-3-mini** (Chat completion)
- model-router** (Chat completion)
- o3** (Chat completion)
- o4-mini** (Chat completion)

Below this, a section titled "Explore more capabilities" features a large image of a lion's face with the text "Boundless innovation". A call-to-action button "Go to video playground" is overlaid on the image. Two boxes below the image provide links to "Build apps with code templates" and "Explore Azure AI Services". A green arrow points from the "Explore Azure AI Services" box towards the "Explore" icon in the bottom right corner of the page.

Latest models

DeepSeek-R1-0528 Chat completion

grok-3 Chat completion

grok-3-mini Chat completion

model-router Chat completion

o3 Chat completion

o4-mini Chat completion

Explore more capabilities

Boundless innovation

Drive efficiency and engagement with transformative workflows and cutting-edge videos using Sora in Azure AI Foundry Models.

Go to video playground

Build apps with code templates

Use code templates to quickly create a proof-of-concept app and deploy it to production.

Explore Azure AI Services

Create market-ready AI applications using customizable APIs and models.

AI Services - Azure AI Foundry Azure-AI-Content-Understanding

<https://ai.azure.com/explore/aiservices?tid=4e1ee3db-...>

MBI BIM AI PennDOT GIS Azure OneDrive Udemy Azure DevOps WorkshopPLUS

 **Speech**
Enhance customer experiences through speech to text, text to speech, and speech translation features.
[View all Speech capabilities](#)

 **Language + Translator**
Analyze, summarize and translate using LLM-powered natural language processing capabilities.
[View all Language + Translator capabilities](#)

 **Vision + Document**
Discover information and insights from documents, images and video with OCR and multi-modal AI.
[View all Vision + Document capabilities](#)

 **Content Safety**
Detect harmful, offensive, or inappropriate user-generated or AI-generated content in your app including text, image, and multi-modal APIs.
[View all Content Safety capabilities](#)

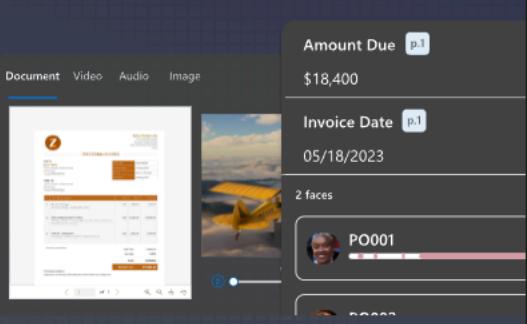


Azure AI Foundry / AI Services / Vision + Document

[Try Content Understanding](#)

Content Understanding

Turn unstructured documents, images, video, and audio into structured data with the new Generative AI-powered Content Understanding service.



[View all other vision capabilities](#)

Module assessment

1. What kinds of AI solution is Azure AI Content Understanding designed to help you build? **Analyzers that extract information from documents, images, videos, and audio files.**
2. Which graphical tool should you use to create an Azure AI Content Understanding project? **Azure AI Foundry portal.**
3. What should you define for the information you want to extract from content? **A schema**

Summary

Azure AI Content Understanding is a multimodal AI service that enables you to extract information from many different kinds of content.

In this module, you learned how to use the **Azure AI Foundry portal** to create a Content Understanding project and build an **analyzer**.

Note: For more information about Azure AI Content Understanding, see [Azure AI Content Understanding documentation](#).

5.2 Create an Azure AI Content Understanding client application

Use the **Azure AI Content Understanding REST API** for multimodal content analysis and information extraction.

Learning objectives

After completing this module, you will be able to:

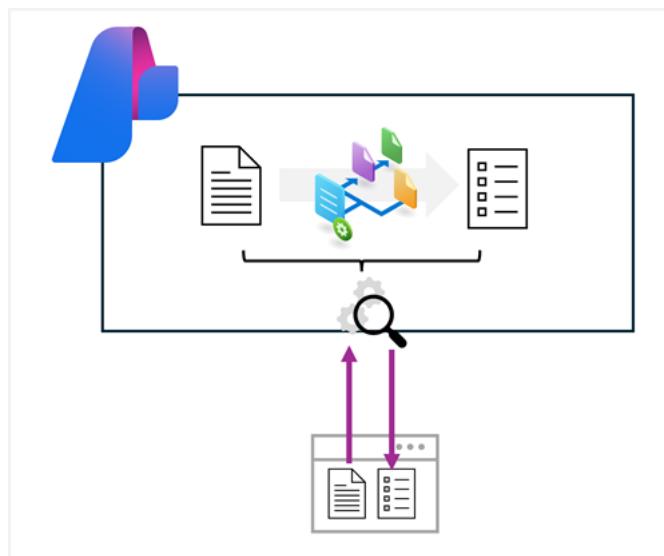
- Use the Azure AI Content Understanding REST API to **build a content analyzer**.
- Use the Azure AI Content Understanding REST API to **consume a content analyzer**.

Introduction

Azure AI Content Understanding is a multimodal service that simplifies the creation of AI-powered analyzers that can extract information from **multiple content formats**, including *documents, images, audio files, and videos*.

Tip: To learn how to build Azure AI Content Understanding analyzers, complete the [Create a multimodal analysis solution with Azure AI Content Understanding](#) module.

You can develop client applications that use **Azure AI Content Understanding analyzers** by using the **Azure AI Content Understanding REST API**; which is the focus of this module.



In this module, you'll learn how to write code that **uses the REST API to submit a content file to an analyzer and process the results**.

Note: Azure AI Content Understanding is currently in public preview. Details described in this module are subject to change.

Prepare to use the AI Content Understanding REST API

Before you can use the **Azure AI Content Understanding REST API**, you need an **Azure AI services multi-services resource** in your Azure subscription. You can provision this resource in the following ways:

- Create an **Azure AI services** resource in the Azure portal.
- Create an **Azure AI Foundry** hub, which includes an Azure AI services resource by default.

Tip: Creating an Azure AI Foundry hub enables you to work in an Azure AI Foundry project, in which you can use visual tools to create and manage Azure AI Content Understanding schemas and analyzers.

After you've provisioned an Azure AI services resource, you need the following information to connect to the **Azure AI Content Understanding REST API** from a client application:

- The Azure AI services resource **endpoint**
- One of the **API keys** associated with the endpoint.

You can obtain these values from the Azure portal, as shown in the following image:

The screenshot shows the Microsoft Azure portal interface. The left sidebar navigation bar is visible, with 'Keys and Endpoint' selected under the 'Resource Management' section. The main content area displays two API keys, 'KEY 1' and 'KEY 2', each represented by a redacted string of characters. Below the keys is a 'Location/Region' dropdown set to 'eastus'. At the bottom of the page, there is a navigation bar with tabs for 'Content Understanding' (which is highlighted), 'OpenAI', 'Speech', 'Content Safety', 'Computer Vision', and 'Content Understanding' again. A note below the tabs says, 'Use the below endpoints to call into Content Understanding APIs. [Learn more](#)' followed by the URL 'https://my-ai-svcs.services.a...'. The browser address bar at the top shows the URL 'https://ms.portal.azure.com/'.

If you're working within an Azure AI Foundry project, you can find the **endpoint** and **key** for the associated *Azure AI services resource* in the *Azure AI Foundry portal*, as shown in the following image:

The screenshot shows the Azure AI Foundry interface for a 'content-understanding' project. On the left, there's a sidebar with various options like Overview, Model catalog, Playgrounds, AI Services, Agents, Templates, Fine-tuning, Content Understanding (selected), Prompt flow, Assess and improve, Tracing, Evaluation, Safety + security, My assets (Models + endpoints, Data + indexes, Web apps), and Management center. The main content area has tabs for 'Endpoints and keys' and 'Project details'. Under 'Endpoints and keys', there's an 'API Key' field containing a redacted key, 'Included capabilities' (Azure AI inference, Azure OpenAI, Azure AI Services selected), and 'Azure AI Services endpoint' set to https://xxxxxxxxxxxxxxxxxxxxxx.cognitiveservices.azure.com/. It also shows 'Speech to text endpoint' at https://westus.stt.speech.microsoft.com/ and 'Text to speech endpoint' at https://westus.tts.speech.microsoft.com/. There's a link to 'API documentation' and a 'View all endpoints' button. Under 'Project details', there's a 'Project connection string' (redacted) and a 'Subscription' section showing 'My Subscription'. A 'Help' button is in the top right.

When working in an Azure AI Foundry project, you can also write code that uses the **Azure AI Foundry SDK** to connect to the project using **Microsoft Entra ID authentication**, and retrieve the connection details for the Azure AI services resource; including the endpoint and key.

Tip: To learn more about programming with the Azure AI Foundry SDK, complete the [Develop an AI app with the Azure AI Foundry SDK](#) module.

Create a Content Understanding analyzer

In most scenarios, you should consider creating and testing analyzers using the **visual interface in the Azure AI Foundry portal**. However, in some cases you might want to create an analyzer by submitting a **JSON definition of the schema** for your desired content fields to the **REST API**.

Defining a schema for an analyzer

Analyzers are based on **schemas that define the fields you want to extract or generate from a content file**. At its simplest, a schema is a set of fields, which can be specified in a JSON document, as shown in this example of an analyzer definition:

```
{ "description": "Simple business card", "baseAnalyzerId": "prebuilt-documentAnalyzer", "config": { }, "returnDetails": true }
```

This example of a custom analyzer schema is based on the **pre-built document analyzer**, and describes *two fields* that you would expect to find on a business card: *ContactName* and *EmailAddress*. Both fields are defined as string data types, and are expected to be extracted from a document (in other words, the string values are expected to exist in the document so they can be "read"; rather than being fields that can be generated by inferring information about the document).

Note: This example is deliberately simple, with the minimal information needed to create a working analyzer. In reality, the schema would likely include more fields of different types, and the analyzer definition would include more configuration settings. The JSON might even include a sample document. See the [Azure AI Content Understanding REST API documentation](#) for more details.

Using the REST API to create an analyzer

With your analyzer definition in place, you can use the REST API to submit it to Azure AI Content Understanding to be created. The JSON data is submitted as a `PUT` request to the **endpoint** with the **API key in the request header** to start the analyzer creation operation.

The response from the `PUT` request includes a `Operation-Location` in the header, which provides a **callback URL** that you can use to **check on the status of the request** by submitting a `GET` request.

You can use any HTTP-capable client tool or language to submit the request. For example, the following Python code submits a request to create an analyzer based on the contents of a file named `card.json` (which is assumed to contain the JSON definition described previously)

```
import json
import requests

# Get the business card schema
with open("card.json", "r") as file:
    schema_json = json.load(file)

# Use a PUT request to submit the schema for a new analyzer
analyzer_name = "business_card_analyser"

headers = {
    "Ocp-Apim-Subscription-Key": "<YOUR_API_KEY>",
    "Content-Type": "application/json"}

url = f"<YOUR_ENDPOINT>/contentunderstanding/analyzers/{analyzer_name}?api-version=2025-05-01-preview"

response = requests.put(url, headers=headers, data=json.dumps(schema_json))

# Get the response and extract the ID assigned to the operation
callback_url = response.headers["Operation-Location"]

# Use a GET request to check the status of the operation
result_response = requests.get(callback_url, headers=headers)

# Keep polling until the operation is complete
status = result_response.json().get("status")
while status == "Running":
    result_response = requests.get(callback_url, headers=headers)
    status = result_response.json().get("status")

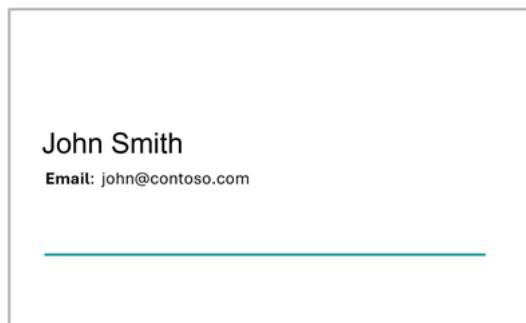
print("Done!")
```

Analyze content

To analyze the contents of a file, you can use the **Azure AI Content Understanding REST API** to submit it to the **endpoint** using a `POST` request. You can specify the **content as a URL** (for a file hosted in an Internet-accessible location) or as the **binary contents of the file** (for example, a .pdf document, a .png image, an .mp3 audio file, or an .mp4 video file). The request header must include the **API key**, and the **endpoint** address for the analyze request includes the analyzer to be used.

As with the request to create an analyzer, the analyze request starts an **asynchronous** operation. The `POST` request returns a unique operation ID, which you can then use in a `GET` request to **check the status of the analysis operation**.

For example, suppose you want to use the *business card analyzer* discussed previously to extract the name and email address from the following scanned business card image:



The following Python code submits a request for analysis, and then polls the service until the operation is complete and the results are returned.

```

1 import json
2 import requests
3
4 # Read the image data
5 with open("business-card.png", "rb") as file:
6     image_data = file.read()
7
8 ## Use a POST request to submit the image data to the analyzer
9 analyzer_name = "business_card_analyser"
10
11 headers = {
12     "Ocp-Apim-Subscription-Key": "<YOUR_API_KEY>",
13     "Content-Type": "application/octet-stream"}
14
15 url = f"{<YOUR_ENDPOINT>}/contentunderstanding/analyzers/{analyzer_name}:analyze?
api-version=2025-05-01-preview"
16
17 response = requests.post(url, headers=headers, data=image_data)
18
19 # Get the response and extract the ID assigned to the analysis operation
20 response_json = response.json()
21 id_value = response_json.get("id")
22
23 # Use a GET request to check the status of the analysis operation
24 result_url = f"{<YOUR_ENDPOINT>}/contentunderstanding/analyzerResults/{id_value}?
api-version=2025-05-01-preview"
25
26 result_response = requests.get(result_url, headers=headers)
27
28 # Keep polling until the analysis is complete
29 status = result_response.json().get("status")
30 while status == "Running":
31     result_response = requests.get(result_url, headers=headers)
32     status = result_response.json().get("status")
33
34 # Get the analysis results
35 if status == "Succeeded":
36     result_json = result_response.json()

```

Processing analysis results

The results in the response JSON depend on:

- The **kind of content** the *analyzer* is designed to analyze (for example, document, video, image, or audio).
- The **schema** for the analyzer.
- The **contents of the file** that was analyzed.

For example, the response from the document-based business card analyzer when analyzing the business card described previously contain:

- The extracted fields
- The *optical character recognition (OCR)* layout of the document, including locations of lines of text, individual words, and paragraphs on each page.

Here's the complete JSON response for the business card analysis:

- Confidence score
- Span
- Content
- Source

Your application must typically parse the JSON to retrieve field values. For example, the following python code extracts all of the string values:

```

# (continued from previous code example)

# Iterate through the fields and extract the names and type-specific values
contents = result_json["result"]["contents"]
for content in contents:
    if "fields" in content:
        fields = content["fields"]
        for field_name, field_data in fields.items():
            if field_data['type'] == "string":
                print(f"{field_name}: {field_data['valueString']}")

```

The output from this code is shown here:

```
ContactName: John Smith  
EmailAddress: john@contoso.com
```

Exercise - Develop a Content Understanding client application

Now it's your turn to build your own Content Understanding client application!

In this exercise, you use the Azure AI Content Understanding REST API to extract information from content by submitting a file to an analyzer.

Develop a Content Understanding client application

In this exercise, you use Azure AI Content Understanding to create an analyzer that extracts information from business cards. You'll then develop a client application that uses the analyzer to extract contact details from scanned business cards.

Module assessment

1. What configuration values are needed to use the Azure AI Content Understanding REST API? **The endpoint and key for the Azure AI service.**
2. What must be specified when calling the analyze method to extract fields from content? **The name of the analyzer.**
3. How are the extracted fields returned? **As type-specific values.**

Summary

Azure AI Content Understanding is a **multimodal AI service** that enables you to **extract information from many different kinds of content**. The REST API for the service enables you to create client applications that analyze content to extract and generate field values.

Note: For more information about Azure AI Content Understanding, see [Azure AI Content Understanding documentation](#).

5.3 Use prebuilt Document intelligence models

Learn what data you can analyze by choosing **prebuilt Forms Analyzer models** and how to deploy these models in a **Document intelligence solution**.

Learning objectives

In this module, you'll learn to:

- Identify business problems that you can solve by using prebuilt models in **Forms Analyzer**.
- Analyze forms by using the **General Document, Read, and Layout models**.
- Analyze forms by using **financial, ID, and tax prebuilt models**.

Introduction

Many forms and documents that businesses handle are common across disparate companies in different sectors. For example, most companies use invoices and receipts. Microsoft **Azure AI Document Intelligence** includes **prebuilt models** so you can handle common document types easily.

Imagine you conduct polls for private companies and political parties. Participants submit their responses as paper forms or as online PDFs. You've decided to deploy **Azure AI Document Intelligence** to streamline data entry. You want to know if you can use the prebuilt models to generate meaningful data from your forms.

In this module, you'll learn about the capabilities of the **prebuilt models in Azure AI Document Intelligence** and how to use them.

Understand prebuilt models

Prebuilt models in Azure AI Document Intelligence enable you to **extract data from common forms and documents without training your own models**.

In your polling company, polling forms are unique to each survey project, but you also use invoices and receipts to record financial transactions and you have many unstructured documents. You want to know how much work is required to extract names, addresses, amounts, and other information from these documents.

Here, you learn how prebuilt models can help you analyze common document types.

What are prebuilt models?

The general approach used in AI solutions is to provide a large quantity of sample data and then train an optimized model by trying different data features, parameters, and statistical treatments. The combination that best predicts the values that interest you constitute the trained model, and you can use this model to predict values from new data.

Many of the forms that businesses use from day to day are of a few common types. For example, most businesses issue or receive invoices and receipts. Any business that has employees in the United States must use the W-2 tax declaration form. Also you often have more general documents that you might want to extract data from. For these cases, Microsoft helps you by providing prebuilt models.

Prebuilt models are already trained on large numbers of their target form type.

If you want to use Document Intelligence to extract data from one of these common forms or documents, you can choose to use a prebuilt model and you don't have to train your own. Because Microsoft trains these models on a large corpus of examples, you can expect them to provide accurate and reliable results when dealing with their intended forms.

Several of the prebuilt models are trained on specific form types:

- **Invoice** model. Extracts common fields and their values from invoices.
- **Receipt** model. Extracts common fields and their values from receipts.
- **US Tax** model. Unified US tax model that can extract from forms such as W-2, 1098, 1099, and 1040.
- **ID document** model. Extracts common fields and their values from US drivers' licenses, European Union IDs and drivers license, and international passports.
- **Business card** model. Extracts common fields and their values from business cards.
- **Health insurance card** model. Extracts common fields and their values from health insurance cards.
- **Marriage certificate**. Extracts information from marriage certificates.
- **Credit/Debit card** model. Extracts common information from bank cards.
- **Mortgage documents**. Extracts information from mortgage closing disclosure, Uniform Residential Loan Application (Form 1003), Appraisal (Form 1004), Validation of Employment (Form 1005), and Uniform Underwriting and Transmittal Summary (Form 1008).
- **Bank statement** model. Extracts account information including beginning and ending balances, transaction details from bank statements.
- **Pay Stub** model. Extracts wages, hours, deductions, net pay, and other common pay stub fields.
- **Check** model. Extracts payee, amount, date, and other relevant information from checks.

The other models are designed to extract values from documents with less specific structures:

- **Read model**. Extracts text and languages from documents.
- **General document model**. Extract text, keys, values, entities, and selection marks from documents.
- **Layout model**. Extracts text and structure information from documents.

Features of prebuilt models

The prebuilt models are designed to extract different types of data from the documents and forms users submit. To select the right model for your requirements, you must understand these features:

- **Text extraction**. All the prebuilt models extract lines of text and words from hand-written and printed text.
- **Key-value pairs**. Many models extract spans of text within a document that identify a label or key and its response or value as key-values pairs. For example, *a typical key might be Weight and its value might be 31 kg*.
- **Entities**. Text that includes common, more complex data structures can be extracted as entities. **Entity types include people, locations, and dates**.
- **Selection marks**. Some models extract spans of text that indicate a choice as selection marks. These **marks include radio buttons and check boxes**.
- **Tables**. Many models can extract tables in scanned forms included the data contained in cells, the numbers of columns and rows, and column and row headings. Tables with merged cells are supported.
- **Fields**. Models trained for a specific form type identify the values of a fixed set of fields. For example, the Invoice model includes **CustomerName** and **InvoiceTotal** fields.

Also consider that prebuilt models are designed for and trained on generic document and form types. If you have an industry-specific or unique form type that you use often, you might be able to obtain more reliable and predictable results by using a custom model. However, **custom models take time to develop because you must invest the time and resources to train them on example forms before you can use it**. *The larger the number of example forms you provide for training, the better the model is at predicting form content accurately*.

Input requirements

The prebuilt models are flexible but you can help them to return accurate and helpful results by **submitting one clear photo or high-quality scan for each document**.

You must also comply with these requirements when you submit a form for analysis:

- The file must be in **JPEG, PNG, BMP, TIFF, or PDF** format. Additionally, the Read model can accept Microsoft Office files.
- The file must be smaller than **500 MB for the standard tier**, and **4 MB for the free tier**.
- Images must have dimensions **between 50 x 50 pixels and 10,000 x 10,000 pixels**.
- PDF documents must have dimensions **less than 17 x 17 inches or A3 paper size**.
- PDF documents must **not be protected with a password**.

Note: If you can, submit *text-embedded PDF files* because they eliminate errors in character recognition.

PDF and TIFF files can have any number of pages but, in the **standard tier**, only the **first 2,000 pages** are analyzed. In the **free tier**, only the **first two pages** are analyzed.

Try out prebuilt models with Azure AI Document Intelligence Studio

Azure AI Document Intelligence is designed as a web service you can call using code in your custom applications. However, it's often helpful to explore the models and how they behave with your forms visually. You can perform such experiments by using [Azure AI Document Intelligence Studio](#) and use the experience to help design and write your code.

You can choose any of the prebuilt models in Azure AI Document Intelligence Studio. Microsoft provides some sample documents for use with each model or you can add your own documents and analyze them.

The screenshot shows the Azure AI Form Recognizer Studio interface. At the top, there's a blue header bar with the title 'Applied AI | Form Recognizer Studio - Preview' and various icons. Below the header, a navigation bar includes links for 'Help us improve Form Recognizer', 'Take our survey!', 'New support request', and a close button. The main content area has a breadcrumb trail 'Form Recognizer Studio > Prebuilt'. On the left, there's a sidebar with a '+ Add' button, a dropdown menu, and a preview thumbnail of a business card labeled 'bizcard.jpg'. The main workspace displays a business card with the name 'Chris Smith' and contact information. To the right, a results panel titled 'Fields' lists the extracted data fields and their confidence scores. The results are as follows:

Field	Result	Code
DocType	businessCard	
Addresses	#1 4001 1st Ave NE Redmond, WA 98052	95.80%
CompanyNames	#1 CONTOSO	94.90%
ContactNames	#1 Chris Smith	97.80%
Content		98.20%
FirstName		98.20%
Chris		
LastName		98.50%
Smith		
Departments	#1 Cloud & AI Department	84.20%
Emails	#1	

Calling prebuilt models by using APIs

Because **Azure AI Document Intelligence implements RESTful web services**, you can use web service calls from any language that supports them. However, when you use Microsoft's Azure AI Document Intelligence APIs, security and session management is simplified and you have to write less code.

Azure AI Document Intelligence is available for:

- C# and other .NET languages.

- Java.
- Python.
- JavaScript.

Whenever you want to call Azure AI Document Intelligence, you must start by **connecting and authenticating with the service in your Azure subscription**. To make that connection, you need:

- The service **endpoint**. This value is the URL where the service is published.
- The **API key**. This value is a unique key that grants access.

You obtain both of these values from the Azure portal.

Because the service can take a few seconds to respond, it's best to use **asynchronous** calls to submit a form and then obtain results from the analysis:

```
poller = document_analysis_client.begin_analyze_document("prebuilt-layout", AnalyzeDocumentRequest(url_source=docUrl))
result: AnalyzeResult = poller.result()
```

The details you can extract from these results depend on the model you used.

Learn more

- [What is Azure AI Document Intelligence?](#)
- [Azure AI Document Intelligence models](#)

Use the General Document, Read, and Layout models

If you want to extract text, languages, and other information from documents with unpredictable structures, you can use the **read, general document, or layout models**.

In your polling company, customers and partners often send specifications, tenders, statements of work, and other documents with unpredictable structures. You want to know if Azure AI Document Intelligence can analyze and extract values from these documents.

Here, you'll learn about the **prebuilt models** that Microsoft provides **for general documents**.

Using the read model

The **Azure AI Document Intelligence read model** extracts printed and handwritten text from documents and images. It's used to provide *text extraction* in all the other prebuilt models.

The **read model** can also **detect the language** that a line of text is written in and **classify whether it's handwritten or printed text**.

Note: The read model supports more languages for printed text than handwritten text. Check the [documentation](#) to see the current list of supported languages.

For multi-page PDF or TIFF files, you can use the `pages` parameter in your request to fix a page range for the analysis.

The read model is ideal if you want to extract words and lines from documents with no fixed or predictable structure.

Using the general document model

The general document model extends the functionality of the read model by adding the **detection of key-value pairs, entities, selection marks, and tables**. The model can extract these values from **structured, semi-structured, and unstructured documents**.

The general document model is the only prebuilt model to support entity extraction. It can recognize entities such as **people, organizations, and dates** and it runs against the whole document, not just key-value pairs. This approach ensures that, **when structural complexity has prevented the model extracting a key-value pair, an entity can be extracted instead**. Remember, however, that sometimes a single piece of text *might return both a key-value pair and an entity*.

The types of entities you can detect include:

- Person . The name of a person.
- PersonType . A job title or role.
- Location . Buildings, geographical features, geopolitical entities.
- Organization . Companies, government bodies, sports clubs, musical bands, and other groups.
- Event . Social gatherings, historical events, anniversaries.
- Product . Objects bought and sold.
- Skill . A capability belonging to a person.
- Address . Mailing address for a physical location.
- Phone number . Dialing codes and numbers for mobile phones and landlines.
- Email . Email addresses.
- URL . Webpage addresses.
- IP Address . Network addresses for computer hardware.
- DateTime . Calendar dates and times of day.
- Quantity . Numerical measurements with their units.

Using the layout model

As well as extracting text, the layout model returns **selection marks** and **tables** from the input image or PDF file. It's a good model to use when you need **rich information about the structure of a document**.

When you digitize a document, it can be at an odd angle. Tables can have complicated structures with or without headers, cells that span columns or rows, and incomplete columns or rows. The layout model can handle all of these difficulties to extract the complete document structure.

For example, each **table cell** is extracted with:

- Its content text.
- The size and position of its bounding box.
- If it's part of a header column.
- Indexes to indicate its row and column position in the table.

Selection marks are extracted with their **bounding box, a confidence indicator, and whether they're selected or not**.

Learn more

- [Language support for Azure AI Document Intelligence](#)
- [Azure AI Document Intelligence read model](#)
- [Azure AI Document Intelligence general document model](#)
- [Azure AI Document Intelligence layout model](#)

Use financial, ID, and tax models

Azure AI Document Intelligence includes some prebuilt models that are trained on common form types. You can use these models to obtain the values of common fields from **invoices, receipts, business cards**, and more.

In your polling company, invoices and receipts are often submitted as photos or scans of the paper documents. Sometimes the scan is poor and the paper is creased or damaged. You want to know if Azure AI Document Intelligence can get this information into your databases more efficiently than manual data entry.

Here, you'll learn about the prebuilt models that handle financial, identity, and tax documents.

Using the invoice model

Your business both issues invoices and receives them from partner organization. There might be many different formats on paper or in digitized forms and some will have been scanned poorly at odd angles or from creased paper.

The invoice model in Azure AI Document Intelligence can handle these challenges and uses the features of the read model to extract text from invoice scans. In addition, it extracts specific fields that are commonly used on invoices including:

- Customer name and reference ID
- Purchase order number
- Invoice and due dates
- Details about the vendor, such as name, tax ID, physical address.
- Similar details about the customer.
- Billing and shipping addresses.
- Amounts such as total tax, invoice total, and amount due.

Invoices also feature lines, usually in a table, each of which is one purchased item. **For each line**, the invoice model identifies details including:

- The description and product code of the product or service invoiced.
- Amounts such as the unit price, the quantity of items, the tax incurred, and the line total.

Using the receipt model

Receipts have similar fields and structures to invoices, but they record amounts paid instead of amounts charged. Azure AI Document Intelligence faces the same challenges of poor scanning or digitization but can reliably identify fields including:

- **Merchant details** such a name, phone number, and address.
- **Amounts** such as receipt total, tax, and tip.
- The **date and time of the transaction**.

As for invoices, receipts often include a table of items, each of which is a product or service purchased. **For each of these lines**, the model recognizes:

- The name of the item.
- The quantity of the item purchased.
- The unit price of the item.
- The total price for that quantity.

Note: In Azure AI Document Intelligence v3.0 and later, the receipt model supports single-page hotel receipt processing. If a receipt is classified as a hotel receipt, the model extracts extra relevant fields such as **arrival and departure dates**.

Using the ID document model

The ID document model is trained to analyze two types of identity document:

- **United States drivers licenses**.
- **International passports**.

Note: Only the biographical pages of passports can be analyzed. Visas and other travel documents are not supported.

The ID document model can extract fields including:

- First and last names.
- Personal information such as sex, date of birth, and nationality.
- The country and region where the document was issued.
- Unique numbers such as the document number and machine readable zone.
- Endorsements, restrictions, and vehicle classifications.

Important: Since much of the data extracted by the ID document model is personal, it is of a sensitive nature and covered by data protection laws in most jurisdictions. Be sure that you have the permission of the individual to store their data and comply with all legal requirements in the way you handle this information.

Using the business card model

Business cards are a popular way to exchange contact information quickly and often include branding, unusual fonts, and graphic design elements. Fields that the business card model can extract include:

- First and last names.
- Postal addresses.
- Email and website addresses.
- Various telephone numbers.

Using other prebuilt models

Azure AI Document Intelligence offers several prebuilt models, with new models being released regularly. Before training a custom model, it's worth verifying if your use case can be analyzed accurately with one of these prebuilt models. Using a prebuilt model will benefit from rigorous testing, updated model versions, and reduced cost compared to a custom model.

Learn more

- [Azure AI Document Intelligence model overview](#)

Exercise - Analyze a document using Azure AI Document Intelligence

Use prebuilt Document Intelligence models

In this exercise, you'll set up an Azure AI Document Intelligence resource in your Azure subscription. You'll use both the Azure AI Document Intelligence Studio and C# or Python to submit forms to that resource for analysis.

Module assessment

1. You have a large set of documents with varying structures that contain customer name and address information. You want to extract entities for each customer. Which prebuilt model should you use? **General document model**.
2. You are using the prebuilt layout model to analyze a document with many checkboxes. You want to find out whether each box is checked or empty. What object should you use in the returned JSON code? **Selection marks**.
3. You submit a Word document to the Azure AI Document Intelligence general document model for analysis but you receive an error. The file is A4 size, contains 1 MB of data, and is not password-protected. How should you resolve the error? **Convert the document to PDF format**.

Summary

There are many document types that are common to most business and Azure AI Document Intelligence includes prebuilt models to handle them. If you have a collection of such forms that you want to analyze, you can extract data by using these prebuilt models and you don't have to train your own models. You can get up and running very quickly by submitting photos and scans to the most appropriate prebuilt model.

Now that you've completed this module, you can:

- Identify business problems that you can solve by using prebuilt models in Azure AI Document Intelligence.
- Analyze forms by using the General Document, Read, and Layout models.
- Analyze forms by using financial, ID, and tax prebuilt models.

Learn more

- [What is Azure AI Document Intelligence?](#)
- [Azure AI Document Intelligence models](#)
- [Language support for Azure AI Document Intelligence](#)
- [Azure AI Document Intelligence read model](#)
- [Azure AI Document Intelligence general document model](#)
- [Azure AI Document Intelligence layout model](#)
- [Azure AI Document Intelligence invoice model](#)
- [Azure AI Document Intelligence receipt model](#)
- [Azure AI Document Intelligence ID document model](#)
- [Azure AI Document Intelligence business card model](#)
- [Azure AI Document Intelligence W-2 model](#)

5.4 Extract data from forms with Azure Document intelligence

Document intelligence **uses machine learning technology to identify and extract key-value pairs and table data from form documents** with accuracy, at scale. This module teaches you how to use the **Azure Document intelligence cognitive service**.

Learning objectives

In this module, you learn how to:

- Identify how **Document intelligence's layout service, prebuilt models, and custom models** can automate processes.
- Use Document intelligence's capabilities with **SDKs, REST API, and Document Intelligence Studio**.
- Develop and test custom models.

Introduction

Forms are used to communicate information in every industry, every day. Many people still manually **extract data from forms** to exchange information.

Consider some of the instances when a person needs to process form data:

- When filing claims
- When enrolling new patients in an online management system
- When entering data from receipts to an expense report
- When reviewing an operations report for anomalies
- When selecting data from a report to give to a stakeholder

Without AI services, people need to manually sort through form documents to identify important information and then manually reenter data to record it. Some may also need to complete these tasks in real-time with a customer.

Azure Document Intelligence services provide the building blocks for automation by using intelligent services to extract data at scale and with accuracy. **Azure Document Intelligence is a Vision API that extracts key-value pairs and table data from form documents**.

Uses of the Azure Document Intelligence service include:

- Process automation
- Knowledge mining

- Industry-specific applications

In this module, you'll learn how to:

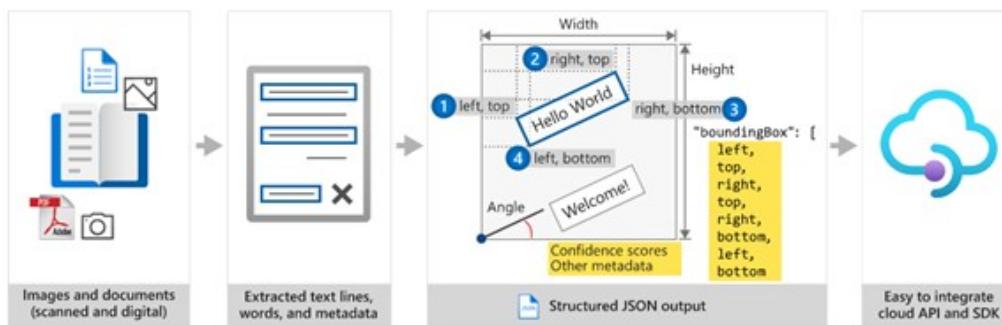
- Identify how Azure Document Intelligence's document analysis, prebuilt, and custom models can automate processes
- Use **Azure Document Intelligence's Optical Character Recognition (OCR) capabilities with SDKs and REST API.**
- Develop and test a custom Azure Document Intelligence model

To complete this module, you'll need a Microsoft Azure subscription. If you don't already have one, you can sign up for a free trial at <https://azure.microsoft.com>.

What is Azure Document Intelligence?

Azure Document Intelligence is one of many Azure AI Services, cloud-based artificial intelligence (AI) services with *REST APIs and client library SDKs* that can be used to build intelligence into your applications.

Azure Document Intelligence uses **Optical Character Recognition (OCR) capabilities and deep learning models** to **extract text, key-value pairs, selection marks, and tables from documents**.



OCR captures document structure by **creating bounding boxes around detected objects in an image**. The locations of the bounding boxes are recorded as coordinates in relation to the rest of the page. Azure Document Intelligence services return bounding box data and other information in a structured form with the relationships from the original file.

Bounding Boxes On Invoice



JSON Response

```
"boundingBox": [136, 139, 351, 138, 351, 166, 136, 166], "text": "Purchase Order", "appearance": { "style": { "name": "other", "confidence": 0.878 }}
```

To build a high-accuracy model from scratch, people need to build deep learning models, use a large amount of compute resources, and face long model training times. These factors could make a project infeasible. Azure Document Intelligence provides underlying models that have been trained on thousands of form examples. The underlying models enable you to do high-accuracy data extraction from your forms with little to no model training.

Azure Document Intelligence service components

Azure Document Intelligence is composed of the following services:

- **Document analysis models:** which *take an input of JPEG, PNG, PDF, and TIFF files and return a JSON file with the location of text in bounding boxes, text content, tables, selection marks (also known as checkboxes or radio buttons), and document structure.*
- **Prebuilt models:** which detect and extract information from document images and return the extracted data in a structured JSON output. Azure Document Intelligence currently supports prebuilt models for several forms, including:
 - W-2 forms
 - Invoices
 - Receipts
 - ID documents
 - Business cards
- **Custom models:** custom models extract data from forms specific to your business. Custom models can be trained through the Azure Document Intelligence Studio.

Note: Some Azure Document Intelligence features are in preview, as of the time this content was authored, and as a result, features and usage details might change. You should refer to the [official page](#) for up-to-date information.

Access services

You can access Azure Document Intelligence services in several ways. These options include using:

- A **REST API**
- Client library **SDKs**
- **Azure Document Intelligence Studio**
- **Azure AI Foundry**

Tip: This module's exercise focuses on the Python and .NET SDKs. The underlying REST services can be used by any language.

Check out the [documentation](#) for quick start guides on all the available SDKs and the REST API.

Get started with Azure Document Intelligence

To start a project with Azure Document Intelligence services, you need an Azure resource and selection of form files for data extraction.

Subscribe to a resource

You can access Azure Document Intelligence services via:

- An **Azure AI Service resource**: a *multi-service subscription key* (used across multiple Azure AI Services)
- OR
- An **Azure Document Intelligence resource**: a *single-service subscription key* (used only with a specific Azure AI Service)

Note: Create an **Azure AI Services resource** if you plan to access multiple Azure AI services **under a single endpoint/key**. For Azure Document Intelligence access only, create an Azure Document Intelligence resource.

Note that **you need a single-service resource if you intend to use Microsoft Entra authentication**.

You can subscribe to a service in the Azure portal or with the Azure Command Line Interface (CLI). You can learn more about the CLI commands [here](#).

Understand Azure Document Intelligence file input requirements

Azure Document Intelligence works on input documents that meet these requirements:

- Format must be **JPG, PNG, BMP, PDF (text or scanned), or TIFF**.
- The file size must be **less than 500 MB** for paid (S0) tier and **4 MB** for free (F0) tier.
- Image dimensions must be **between 50 x 50 pixels and 10,000 x 10,000 pixels**.
- The total size of the training data set must be **500 pages or less**.

More input requirements can be found in the [documentation](#) for specific models.

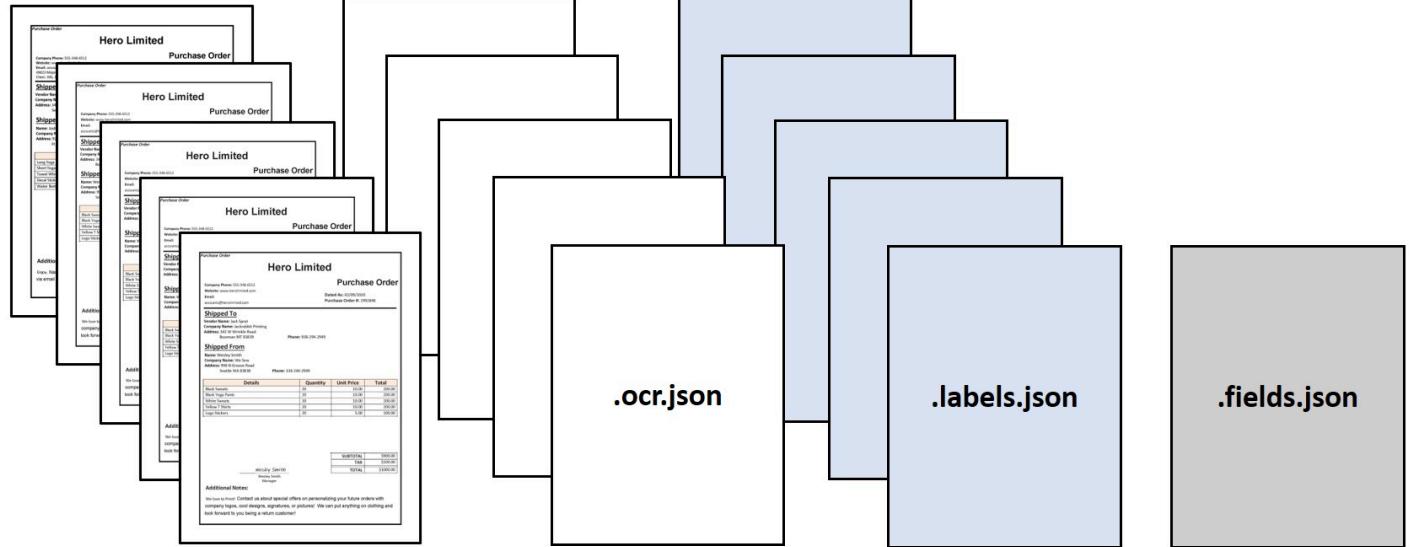
Decide what component of Azure Document Intelligence to use

After you collect your files, decide what you need to accomplish.

Use case	Recommended features to use
Use OCR capabilities to capture document analysis	Use the Layout model, Read model, or General Document model .
Create an application that extracts data from W-2s, Invoices, Receipts, ID documents, Health insurance, vaccination, and business cards	Use a prebuilt model . These models don't need to be trained. Azure Document Intelligence services analyze the documents and return a JSON output .
Create an application to extract data from your industry-specific forms	Create a custom model . This model needs to be trained on sample documents. After you train the custom model, it can analyze new documents and return a JSON output .

Train custom models

Azure's **Azure Document Intelligence service supports supervised machine learning**. You can train custom models and create composite models with form documents and JSON documents that contain labeled fields.



Examples of files needed for supervised (labeled) custom form model training

To train a custom model:

- Store sample forms in an **Azure blob container**, along with **JSON files containing layout and label field information**.
 - You can generate an **ocr.json** file for each sample form using the Azure Document Intelligence's Analyze document function. Additionally, you need a single **fields.json** file describing **the fields you want to extract**, and a **labels.json** file for **each sample form mapping the fields to their location in that form**.
- Generate a **shared access security (SAS) URL** for the container.
- Use the **Build model REST API function (or equivalent SDK method)**.
- Use the **Get model REST API function (or equivalent SDK method)** to get **the trained model ID**.

OR

- Use the Azure Document Intelligence Studio to label and train. There are two types of underlying models for custom forms: custom template models or custom neural models.
 - **Custom template models** accurately extract **labeled key-value pairs, selection marks, tables, regions, and signatures from documents**. Training only takes a few minutes, and more than 100 languages are supported.
 - **Custom neural models** are deep learned models that combine layout and language features to accurately **extract labeled fields from documents**. This model is **best for semi-structured or unstructured documents**.

Use Azure Document Intelligence models

Using the API

To extract form data using a custom model, use the analyze document function of either a **supported SDK, or the REST API**, while **supplying model ID** (generated during model training). This function starts the form analysis, which you can then request the result to get the analysis.

Example code to call your model:

```
endpoint = "YOUR_DOC_INTELLIGENCE_ENDPOINT"
key = "YOUR_DOC_INTELLIGENCE_KEY"

model_id = "YOUR_CUSTOM_BUILT_MODEL_ID"
formUrl = "YOUR_DOCUMENT"

document_analysis_client = DocumentAnalysisClient(    endpoint=endpoint, credential=AzureKeyCredential(key) )

# Make sure your document's type is included in the list of document types the custom model can analyze
task = document_analysis_client.begin_analyze_document_from_url(model_id, formUrl)
result = task.result()
```

A successful JSON response contains **analyzeResult** that contains **the content extracted** and **an array of pages containing information about the document content**.

Example analyze document JSON response:

Explore the documentation for [supported language quickstarts](#).

Understanding confidence scores

If the confidence values of the **analyzeResult** are low, try to improve the quality of your input documents.

You want to make sure that the form you're analyzing has a similar appearance to forms in the training set if the confidence values are low. If the form appearance varies, **consider training more than one model, with each model focused on one form format**.

Depending on the use case, you might find that **a confidence score of 80% or higher is acceptable for a low-risk application. For more sensitive cases, like reading medical records or billing statements, a score of 100% is recommended**.

Use the Azure Document Intelligence Studio

In addition to SDKs and the REST API, **Azure Document Intelligence** services can be accessed through a user interface called the **Azure Document Intelligence Studio**, an online tool for visually exploring, understanding, and

integrating features from the Azure Document Intelligence service. The Studio can be used to **analyze form layouts, extract data from prebuilt models, and train custom models**.

Azure AI | Document Intelligence Studio

① Azure Form Recognizer is now Azure AI Document Intelligence. [Learn more](#) about the latest updates to the service and the Studio experience.

Form Recognizer Studio

Get started with Document Intelligence Studio

Extract text, key-value pairs, tables, and structures from forms and documents using common layouts and prebuilt models, or create your own custom models. [Learn more](#)

Document analysis

Extract text, tables, structure, key-value pairs, and named entities from documents.



Read
Extract printed and handwritten text along with barcodes, formulas and font styles from images and documents.
[Try it out](#)



Layout
Extract tables, check boxes, and text from forms and documents.
[Try it out](#)



General documents
Extract key value pairs and structure like tables and selection marks from any form or document.
[Try it out](#)

Prebuilt models

Extract data from unique document types using the following prebuilt models.



Invoices
Extract invoice ID, customer details, vendor
[Try it out](#)



Receipts
Extract time and date of the transaction,
[Try it out](#)



Business cards
Extract person name, job title, address,
[Try it out](#)



Identity documents
Extract name, expiration date, machine
[Try it out](#)

The Azure Document Intelligence Studio currently supports the following projects:

- **Document analysis models**
 - **Read:** Extract **printed and handwritten text lines, words, locations, and detected languages from documents and images.**
 - **Layout:** Extract **text, tables, selection marks, and structure information from documents (PDF and TIFF) and images (JPG, PNG, and BMP).**
 - **General Documents:** Extract **key-value pairs, selection marks, and entities from documents.**
- **Prebuilt models**
- **Custom models**

Build Document analysis model projects

To **extract text, tables, structure, key-value pairs, and named entities** with **document analysis models**:

- Create an Azure Document Intelligence or Azure AI Services resource.

- Select either "**Read**", "**Layout**", or "**General Documents**" under the Document analysis models category.
- Analyze your document. You'll need your Azure Document Intelligence or Azure AI service **endpoint and key**.

Build prebuilt model projects

To extract data from common forms with prebuilt models:

- Create an **Azure Document Intelligence** or **Azure AI Services** resource
- Select one of the "**prebuilt models**" including **W-2s, Invoices, Receipts, ID documents, Health insurance, vaccination, and business cards**.
- Analyze your document. You'll need your Azure Document Intelligence or Azure AI service **endpoint and key**.

Build custom model projects

You can use **Azure Document Intelligence Studio's custom service** for the entire process of training and testing custom models.

When you use Azure Document Intelligence Studio to build custom models, the **ocr.json** files, **labels.json** files, and **fields.json** file needed for training are automatically created and stored in your storage account.

To train a custom model and use it to extract data with custom models:

- Create an Azure Document Intelligence or Azure AI Services resource
- Collect **at least 5-6 sample forms** for training and upload them to your storage account container.
- Configure **cross-domain resource sharing (CORS)**. CORS enables Azure Document Intelligence Studio to store labeled files in your storage container.
- **Create a custom model project in Azure Document Intelligence Studio**. You'll need to provide configurations linking your storage container and Azure Document Intelligence or Azure AI Service resource to the project.
- Use Azure Document Intelligence Studio to **apply labels to text**.
- Train your model. Once the model is trained, you'll receive a **Model ID and Average Accuracy for tags**.
- Test your model by analyzing a new form that wasn't used in training.

Exercise - Extract data from custom forms

In this exercise, you'll use the Azure Document Intelligence service to train and test a custom model using the Python or .NET SDK.

To complete the exercise for this module, launch the VM and follow the instructions.

Extract Data from Forms

Suppose a company currently requires employees to manually purchase order sheets and enter the data into a database. They would like you to utilize AI services to improve the data entry process. You decide to build a machine

learning model that will read the form and produce structured data that can be used to automatically update a database.

Azure AI Document Intelligence is an Azure AI service that enables users to build automated data processing software. This software can **extract text, key/value pairs, and tables from form documents using optical character recognition (OCR)**. Azure AI Document Intelligence has **pre-built models for recognizing invoices, receipts, and business cards**. The service also provides the capability to train **custom models**. In this exercise, we will focus on building custom models.

Module assessment

In this module, you have learned how to use the Azure Document Intelligence service to extract data from forms.

Consider the following review questions to check your understanding of the topics discussed in this module.

1. A person plans to use an Azure Document Intelligence prebuilt invoice model. To extract document data using the model and REST API language, what are two calls they need to make to the API? **Analyze Invoice and Get Analyze Invoice Result**.
2. A person needs to build an application that submits expense claims and extracts the merchant, date, and total from scanned receipts. What's the best way to build the application? **Use Azure Document Intelligence's prebuilt receipts model**.
3. A person is building a custom model with Azure Document Intelligence services. What is required to train a model? **Along with the form to analyze, JSON files need to be provided**.

Summary

This module focused on Azure Document Intelligence's **prebuilt service, custom service, and its client library SDKs and REST API**. We also introduced the **Azure Document Intelligence Studio** to label and train your model.

Azure Document Intelligence services can be integrated with other Azure AI Services. For example, you can try using [this tutorial](#) with **Azure Document Intelligence and Cognitive Search**.

Document intelligence is just one part of the overall Vision API in Azure AI Services. Azure Document Intelligence services are ever evolving. You can read about the latest updates [here](#).

5.5 Create a knowledge mining solution with Azure AI Search

Unlock the hidden insights in your data with **Azure AI Search**. In this module, you'll learn how to implement a **knowledge mining solution** that extracts and enriches data, making it searchable and ready for deeper analysis.

Learning objectives

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

- Implement **indexing** with **Azure AI Search**
- Use AI skills to **enrich data in an index**
- Search an index to find relevant information
- Persist extracted information in a knowledge store

Introduction

Azure AI Search is a powerful cloud-based service that enables you to **extract, enrich, and explore information** from a wide variety of data sources. In this module, you'll learn how to build **intelligent search and knowledge mining solutions** using Azure AI Search.

We'll start by introducing the core concepts of Azure AI Search, including how to **connect to data sources and create indexes**. You'll discover how the indexing process works, and how AI skills can be used to enrich your data with insights such as **language detection, key phrase extraction, and image analysis**.

After learning how to implement an index, you'll explore how to **query and filter results** using full-text search.

Finally, you'll see how to **use the knowledge store to persist enriched data** for further analysis and integration with other systems.

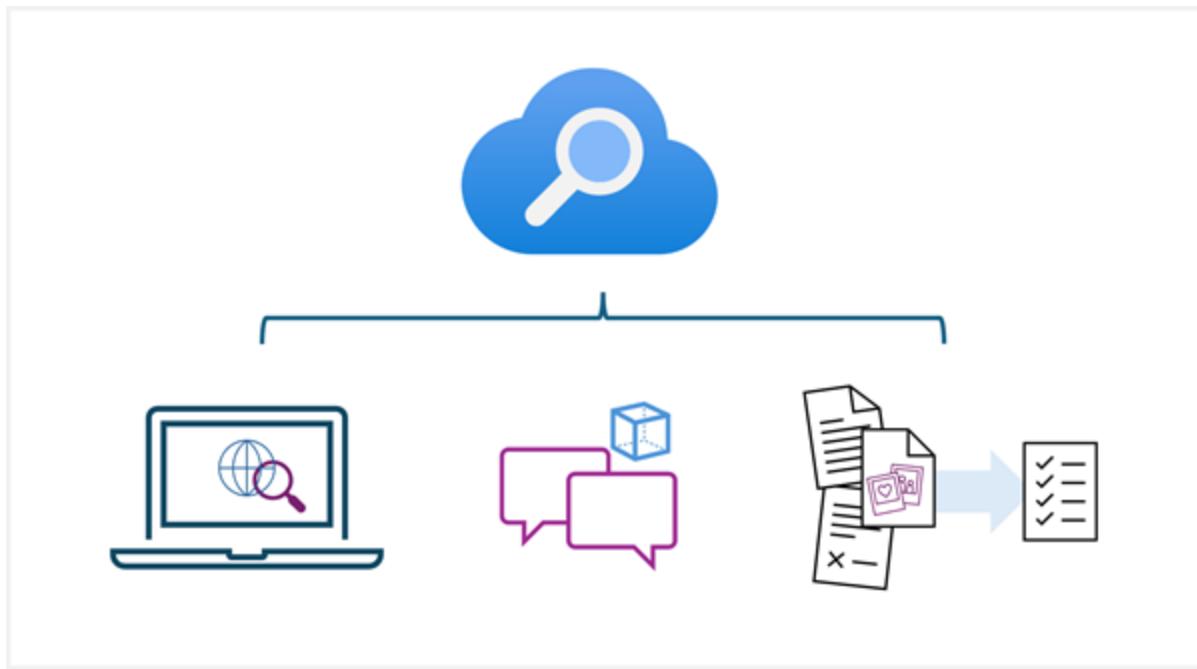
What is Azure AI Search?

Azure AI Search provides a cloud-based solution for **indexing and querying a wide range of data sources**, and **creating comprehensive and high-scale search solutions**. It provides the infrastructure and tools to create search solutions that **extract data from structured, semi-structured, and non-structured documents and other data sources**.

With Azure AI Search, you can:

- **Index documents and data** from a range of sources.
- Use AI skills to **enrich index data**.
- **Store extracted insights in a knowledge store** for analysis and integration.

Azure AI Search indexes contain insights extracted from your data; which can include **text inferred or read using OCR from images, entities and key phrases detection through text analytics**, and other derived information based on AI skills that are integrated into the indexing process.



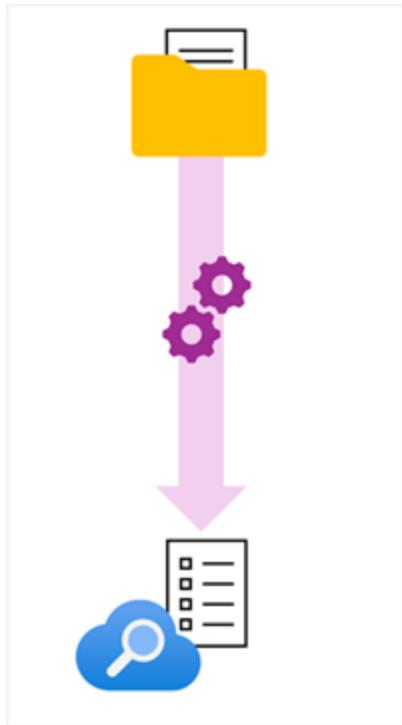
Azure AI search has many applications, including:

- Implementing an enterprise search solution to help employees or customers find information in websites or applications.
- **Supporting retrieval augmented generation (RAG) in generative AI applications** by using vector-based indexes for prompt grounding data.
- Creating **knowledge mining solutions** in which the indexing process is used to infer insights and extract granular data assets from documents to support data analytics.

In this module, we'll focus on Azure AI Search in knowledge mining scenarios.

Extract data with an indexer

At the heart of Azure AI Search solutions is the creation of an index. **An index contains your searchable content and is created and updated, unsurprisingly, by an indexer.**



The indexing process starts with a data source: the storage location of your original data artifacts; for example, an Azure blob store container full of documents, a database, or some other store.

The Indexer automates the extraction and indexing of data fields through an enrichment pipeline, in which it applies *document cracking* to extract the contents of the source documents and applies *incremental steps* to create a hierarchical (JSON-based) document with the required fields for the index definition.

The result is a populated index, which can be queried to return specified fields from documents that match the query criteria.

How documents are constructed during indexing

The indexing process works by creating a document for each indexed entity. *During indexing, an enrichment pipeline iteratively builds the documents that combine metadata from the data source with enriched fields extracted or generated by skills.* You can think of **each indexed document as a JSON**

structure, which initially consists of a document with the index fields you have mapped to fields extracted directly from the source data, like this:

- document
 - metadata_storage_name
 - metadata_author
 - content

When the documents in the data source contain images, you can configure the indexer to extract the image data and place each image in a **normalized_images** collection, like this:

- document
 - metadata_storage_name
 - metadata_author
 - content
 - normalized_images
 - image0
 - image1

Normalizing the image data in this way enables you to use the collection of images as an input for skills that extract information from image data.

Each skill adds fields to the document, so for example a skill that detects the language in which a document is written might store its output in a **language** field, like this:

- document
 - metadata_storage_name
 - metadata_author
 - content
 - normalized_images
 - image0
 - image1
 - language

The document is structured hierarchically, and the skills are applied to a specific context within the hierarchy, enabling you to run the skill for each item at a particular level of the document. For example, you could run an **optical character recognition (OCR)** skill for each image in the normalized images collection to extract any **text** they contain:

- document
 - metadata_storage_name

- metadata_author
- content
- normalized_images
 - image0
 - Text
 - image1
 - Text
- language

The output fields from each skill can be used as inputs for other skills later in the pipeline, which in turn store their outputs in the document structure. For example, we could use a merge skill to combine the original text content with the text extracted from each image to create a new **merged_content** field that contains all of the text in the document, including image text.

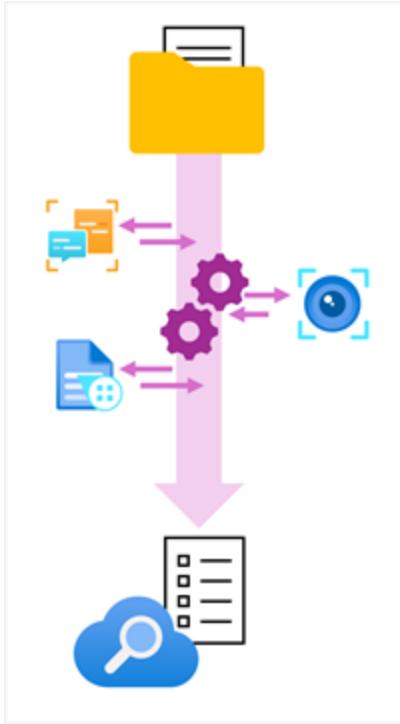
- document
 - metadata_storage_name
 - metadata_author
 - content
 - normalized_images
 - image0
 - Text
 - image1
 - Text
 - language
 - merged_content

The fields in the final document structure at the end of the pipeline are mapped to index fields by the indexer in one of two ways:

- **Fields extracted directly from the source data are all mapped to index fields.** These mappings can be **implicit** (*fields are automatically mapped to index fields with the same name in the index*) or **explicit** (*a mapping is defined to match a source field to an index field, often to rename the field to something more useful or to apply a function to the data value as it is mapped*).
- Output fields from the skills in the skillset are explicitly mapped from their hierarchical location in the output to the target field in the index.

Enrich extracted data with AI skills

The enrichment pipeline that is orchestrated by an indexer uses a skillset of AI skills to create AI-enriched fields. **The indexer applies each skill in order, refining the index document at each step.**



Built-in skills

Azure AI Search provides a collection of built-in skills that you can include in a skillset for your indexer.

Built-in skills include functionality from Azure AI services such as Azure AI Vision and Azure AI Language, enabling you to apply enrichments such as:

- Detecting the **language** that text is written in.
- Detecting and extracting **places, locations, and other entities in the text**.
- Determining and extracting **key phrases** within a body of text.
- **Translating** text.
- Identifying and extracting (or removing) **personally identifiable information (PII)** within the text.
- Extracting **text from images**.
- Generating **captions and tags** to describe images.

To use the built-in skills, your indexer must have access to an Azure AI services resource. You can use a restricted Azure AI search resource that is included in Azure AI Search (and which is limited to indexing 20 or fewer documents) or you can attach an Azure AI services resource in your Azure subscription (which must be in the same region as your Azure AI Search resource).

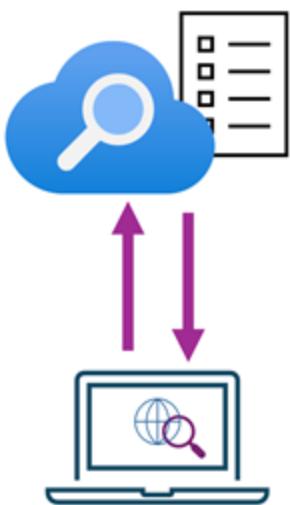
Custom skills

You can further extend the enrichment capabilities of your index by creating custom skills. As the name suggests, custom skills perform custom logic on input data from your index document to return new field values that can be incorporated into the index. Often, **custom skills are "wrappers" around services that are specifically designed to extract data from documents**. For example, you could implement a custom skill as an Azure Function, and use it to pass data from your index document to an **Azure AI Document Intelligence** model, which can extract fields from a form.

Tip: To learn more about using custom skills with Azure AI Search, see [Add a custom skill to an Azure AI Search enrichment pipeline](#) in the Azure AI Search documentation.

Search an index

The index is the searchable result of the indexing process. It consists of a collection of JSON documents, with fields that contain the values extracted during indexing. Client applications can query the index to retrieve, filter, and sort information.



Each index field can be configured with the following attributes:

- **key**: Fields that define a unique key for index records.
- **searchable**: Fields that can be queried using full-text search.

- **filterable**: Fields that can be included in filter expressions to return only documents that **match specified constraints**.
- **sortable**: Fields that can be used to **order** the results.
- **facetable**: Fields that can be used to determine values for facets (user interface elements used to filter the results based on a list of **known field values**).
- **retrievable**: Fields that can be included in search results (by default, all fields are retrievable unless this attribute is explicitly removed).

Full-text search

While you could retrieve index entries based on simple field value matching, most search solutions use full-text search semantics to query an index.

Full-text search describes search solutions that parse text-based document contents to find query terms. Full-text search queries in Azure AI Search are based on the [Lucene query syntax](#), which provides a rich set of query operations for searching, filtering, and sorting data in indexes. Azure AI Search supports two variants of the Lucene syntax:

- **Simple** - An intuitive syntax that makes it easy to perform basic searches that match literal query terms submitted by a user.
- **Full** - An extended syntax that supports complex filtering, regular expressions, and other more sophisticated queries.

Client applications submit queries to Azure AI Search by specifying a search expression along with other parameters that determine how the expression is evaluated and the results returned. Some common parameters submitted with a query include:

- **search** - A search expression that includes the terms to be found.
- **queryType** - The [Lucene syntax](#) to be evaluated (simple or full).
- **searchFields** - The index fields to be searched.
- **select** - The fields to be included in the results.
- **searchMode** - Criteria for including results based on multiple search terms. For example, suppose you search an index of travel-related documents for comfortable hotel. A searchMode value of Any returns documents that contain "comfortable", "hotel", or both; while a searchMode value of All restricts results to documents that contain both "comfortable" and "hotel".

Query processing consists of four stages:

1. **Query parsing**. The search expression is *evaluated and reconstructed as a tree of appropriate subqueries*. Subqueries might include **term queries** (finding specific individual words in the

search expression - for example hotel), **phrase queries** (finding multi-term phrases specified in quotation marks in the search expression - for example, "free parking"), and **prefix queries** (finding terms with a specified prefix - for example air*, which would match airway, air-conditioning, and airport).

2. **Lexical analysis** - The query terms are analyzed and refined based on linguistic rules. For example, *text is converted to lower case and nonessential stopwords (such as "the", "a", "is", and so on) are removed*. Then words are *converted to their root form* (for example, "comfortable" might be simplified to "comfort") and *composite words are split into their constituent terms*.
3. **Document retrieval** - The *query terms are matched against the indexed terms*, and the set of matching documents is identified.
4. **Scoring** - A *relevance score* is assigned to each result based on a term [frequency/inverse document frequency \(TF/IDF\) calculation](#).

Tip: For more information about querying an index, and details about simple and full syntax, see [Query types and composition in Azure AI Search](#) in the Azure AI Search documentation.

It's common in a search solution for users to want to refine query results by filtering and sorting based on field values. Azure AI Search supports both of these capabilities through the search query API.

Filtering results

You can apply filters to queries in two ways:

- By including filter criteria in a simple search expression.
- By providing an OData filter expression as a **\$filter** parameter with a *full* syntax search expression.

You can apply a filter to any *filterable* field in the index.

For example, suppose you want to find documents containing the text *London* that have an **author** field value of *Reviewer*.

You can achieve this result by submitting the following simple search expression:

```
search=London+author='Reviewer'  
queryType=Simple
```

Alternatively, you can use an OData filter in a **\$filter** parameter with a **full Lucene search expression** like this:

```
search=London  
$filter=author eq 'Reviewer'
```

```
queryType=Full
```

Note: OData **\$filter** expressions are case-sensitive!

Filtering with facets

Facets are a useful way to present users with filtering criteria based on field values in a result set. They work best when a field has a small number of discrete values that can be displayed as links or options in the user interface.

To use facets, you must specify **facetable** fields for which you want to retrieve the possible values in an initial query. For example, you could use the following parameters to return all of the possible values for the author field:

```
search=*  
facet=author
```

The results from this query include a collection of discrete facet values that you can display in the user interface for the user to select. Then in a subsequent query, you can use the selected facet value to filter the results:

```
search=*  
$filter=author eq 'selected-facet-value-here'
```

Sorting results

By default, results are sorted based on the relevancy score assigned by the query process, with the highest scoring matches listed first. However, you can override this sort order by including an OData **orderby** parameter that specifies one or more **sortable** fields and a **sort order (asc or desc)**.

For example, to sort the results so that the most recently modified documents are listed first, you could use the following parameter values:

```
search=*  
$orderby=last_modified desc
```

Tip: For more information about using filters, see [Filters in Azure AI Search](#) in the Azure AI Search documentation.

Persist extracted information in a knowledge store

While the index might be considered the primary output from an indexing process, the enriched data it contains might also be useful in other ways. For example:

- Since the index is essentially a collection of JSON objects, each representing an indexed record, it might be useful to *export the objects as JSON files for integration into a data orchestration process for extract, transform, and load (ETL) operations.*
- You may want to *normalize the index records into a relational schema of tables* for analysis and reporting.
- Having *extracted embedded images from documents during the indexing process*, you might want to save those images as files.

Azure AI Search supports these scenarios by enabling you to define a **knowledge** store in the skillset that encapsulates your enrichment pipeline. The knowledge store consists of *projections of the enriched data*, which can be *JSON objects, tables, or image files*. When an indexer runs the pipeline to create or update an index, the projections are generated and persisted in the knowledge store.

Tip: To learn more about using a knowledge store, see [Knowledge store in Azure AI Search](#) in the Azure AI Search documentation.

Exercise - Create a knowledge mining solution

It's time to put what you've learned into practice!

In this exercise, you use Azure AI Search to extract and enrich information from documents into a searchable index and a knowledge store.

Create an knowledge mining solution

In this exercise, you use AI Search to index a set of documents maintained by Margie's Travel, a fictional travel agency. The indexing process involves using AI skills to extract key information to make them searchable, and generating a knowledge store containing data assets for further analysis.

Module assessment

1. Which component of an Azure AI Search solution is scheduled to extract and enrich data to populate an index? **Indexer**.
2. Which service supports built-in AI skills in Azure AI Search? **Azure AI Services**
3. Which kind of projection results in a relational data schema for extracted fields? **Table**

Summary

In this module, you've learned how Azure AI Search enables you to build **intelligent search and knowledge mining** solutions by **indexing and enriching data** from various sources. You explored the indexing process, the use of AI skills for data enrichment, and how to persist enriched data in a knowledge store for further analysis and integration.

With these skills, you're now equipped to design and implement solutions that unlock valuable insights from your data using Azure AI Search.

Tip: To learn more about Azure AI Search, see the [Azure AI Search documentation](#).