AI-102

Table of Content

Resources

Microsoft Learning — AI-102

Number	Name of Lab	Url
Lab #1	Create and Consume Azure AI Services	https://microsoftlearning.github.io/mslearn-ai-services/Instructions/Exercises/01-use-azure-ai-services.html
Lab #2	Secure Azure AI Services	https://microsoftlearning.github.io/mslearn-ai-services/Instructions/Exercises/02-ai-services-security.html
Lab #3	Monitoring Azure AI Services	https://microsoftlearning.github.io/mslearn-ai-services/Instructions/Exercises/03-monitor-ai-services.html
Lab #4	Containers	https://microsoftlearning.github.io/mslearn-ai-services/Instructions/Exercises/04-use-a-container.html
Lab #5	Analyze Images	https://microsoftlearning.github.io/mslearn-ai-vision/Instructions/Exercises/01-analyze-images.html
Lab #6	Custom Vision Model	https://microsoftlearning.github.io/mslearn-ai-vision/Instructions/Exercises/02-image-

		classification.html
Lab #7	Detect, Analyze, and Identify Faces	https://microsoftlearning.github.io/mslearn-ai-vision/Instructions/Exercises/04-face-service.html
Lab #8	Optical Character Recognition (OCR)	https://microsoftlearning.github.io/mslearn-ai-vision/Instructions/Exercises/05-ocr.html
Lab #9	Analyze Video	https://microsoftlearning.github.io/mslearn-ai-vision/Instructions/Exercises/06-video-indexer.html
Lab #10	Analyze Text	https://microsoftlearning.github.io/mslearn-ai-language/Instructions/Exercises/01-analyze-text.html
Lab #11	Create a Question Answering Solution	https://microsoftlearning.github.io/mslearn-ai-language/Instructions/Exercises/02-qna.html
Lab #12	Language Understanding Model	https://microsoftlearning.github.io/mslearn-ai-language/Instructions/Exercises/03-language-understanding.html
Lab #13	Custom Text Classification	https://microsoftlearning.github.io/mslearn-ai-language/Instructions/Exercises/04-text-classification.html
Lab #14	Extract Custom Entities	https://microsoftlearning.github.io/mslearn-ai-language/Instructions/Exercises/05-extract-custom-entities.html
Lab #15	Translate Text	https://microsoftlearning.github.io/mslearn-ai-language/Instructions/Exercises/06-translate-text.html
Lab #16	Create a Speech-Enabled App	https://microsoftlearning.github.io/mslearn-ai-language/Instructions/Exercises/07-speech.html
Lab #17	Translate Speech	https://microsoftlearning.github.io/mslearn-ai-language/Instructions/Exercises/08-translate-speech.html
Lab #18	AI Search Solution	https://microsoftlearning.github.io/mslearn-knowledge-mining/Instructions/Exercises/01-azure-

		search.html
Lab #19	Create a Custom Skill for AI Search	https://microsoftlearning.github.io/mslearn-knowledge-mining/Instructions/Exercises/02-search-skills.html
Lab #20	Create a Knowledge Store	https://microsoftlearning.github.io/mslearn-knowledge-mining/Instructions/Exercises/03-knowledge-store.html
Lab #21	Enrich an AI Search Index with Custom Classes	https://microsoftlearning.github.io/mslearn-knowledge-mining/Instructions/Exercises/04-exercise-enrich-cognitive-custom-classes.html
Lab #22	Implement Enhancements to Search Results	https://microsoftlearning.github.io/mslearn-knowledge-mining/Instructions/Exercises/05-exercise-implement-enhancements-to-search-results.html
Lab #23	Enrich a Search Index Using AML	https://microsoftlearning.github.io/mslearn-knowledge-mining/Instructions/Exercises/06-exercise-enrich-search-index-use-model.html
Lab #24	Debug Search Issues	https://microsoftlearning.github.io/mslearn-knowledge-mining/Instructions/Exercises/08-exercise-debug-search-issues.html
Lab #25	Set Up a Semantic Ranker	https://microsoftlearning.github.io/mslearn-knowledge-mining/Instructions/Exercises/09-semantic-search-exercise.html
Lab #26	Run Vector Search Queries	https://microsoftlearning.github.io/mslearn-knowledge-mining/Instructions/Exercises/10-vector-search-exercise.html
Lab #27	Prebuilt Document Intelligence Models	https://microsoftlearning.github.io/mslearn-ai-document-intelligence/Instructions/Exercises/01-use-prebuilt-models.html
Lab #28	Extract Data from Forms	https://microsoftlearning.github.io/mslearn-ai-document-intelligence/Instructions/Exercises/02-custom-document-intelligence.html
Lab #29	Create a Composed	https://learn.microsoft.com/en-us/training/modules/create-composed-form-

	Model	recognizer-model/4-exercise-model
Lab #30	Build and Deploy a Document Intelligence Custom Skill	https://learn.microsoft.com/en-us/training/modules/build-form-recognizer-custom-skill-for-azure-cognitive-search/4-exercise-build-deploy
Lab #31	Get Started with Azure OpenAI Service	https://microsoftlearning.github.io/mslearn-openai/Instructions/Exercises/01-get-started-azure-openai.html
Lab #32	Use Azure OpenAI in your App	https://microsoftlearning.github.io/mslearn-openai/Instructions/Exercises/02-natural-language-azure-openai.html
Lab #33	Generate Images with DALL-E	https://microsoftlearning.github.io/mslearn-openai/Instructions/Exercises/05-generate-images.html
Lab #34	Implement RAG with Azure OpenAI	https://microsoftlearning.github.io/mslearn-openai/Instructions/Exercises/06-use-own-data.html

Azure AI Services

- Vision Studio: https://portal.vision.cognitive.azure.com/
- Azure Video Indexer: https://api-portal.videoindexer.ai/
- **Language:** https://language.cognitive.azure.com/
- Azure AI Search Calculator: https://azure.microsoft.com/en-us/pricing/details/search/

Gen AI Primer

AI vs Generative AI

Artificial Intelligence is computer systems that perform tasks typically requiring human intelligence. In other word, AI's goal is to simulate human intelligence in machines.

These include:

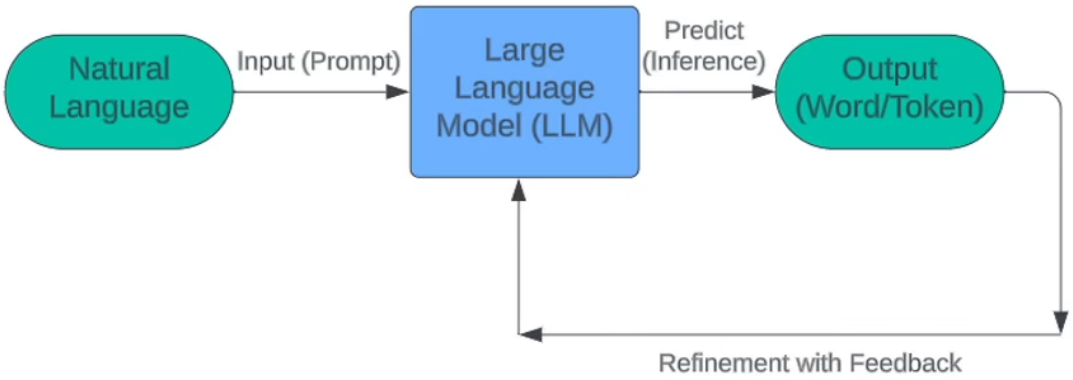
- Problem-solving

- Decision-making
- Understanding natural language
- Recognizing speech and images

Generative AI is a subset of AI that focuses on **creating new content or data** that is novel and realistic. It can interpret or analyze data but also **generates new data itself**.

LLM

- **Large Language Models (LLMs)** are very large deep learning models that are pre-trained on vast amounts of data.
- A LLM is a Foundational Modal that implements the transformer architecture. A Foundational Model is a general-purpose model that is trained on vast amounts of data. FM is pretrained because it can be fined tuned for specific tasks.



Tokenization

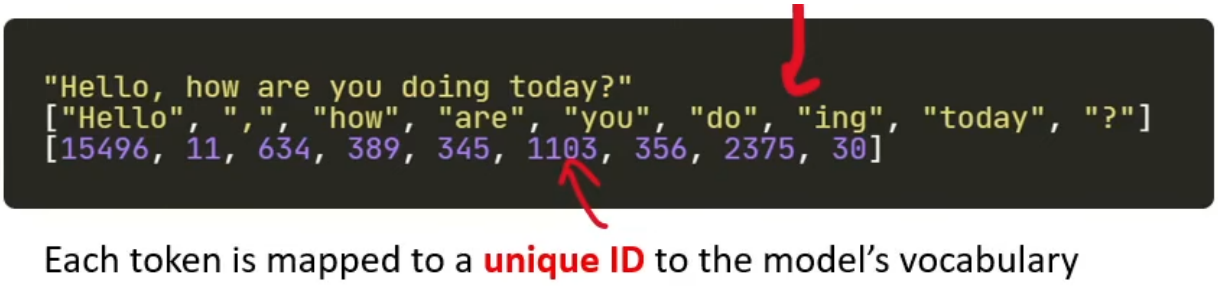
Tokenization is the process of breaking data input (text) into smaller parts.

Some tokenization algorithms are:

- **Byte Pair Encoding (BPE)** used in GPT 3

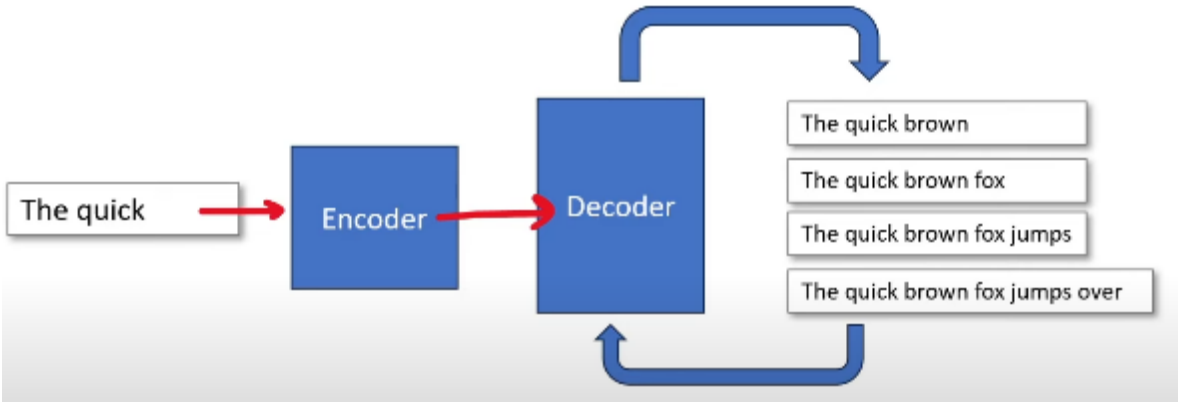
- **WordPiece** used in BERT
- **SentencePiece** used by Google T5 or GPT 3.5

When working with a LLM, the input text must be converted (or 'tokenized') into a sequence of tokens that **match the model's internal vocabulary**.



Tokens and Capacity

When using transformers (in FMs and LLMs), the decoder continuously feeds the sequence of tokens back in as output to help predict the next word in the input.



💡 AI services that offer Models-as-a-Service (MaaS) will often have a limit of combined input and output.

Embeddings

Vector

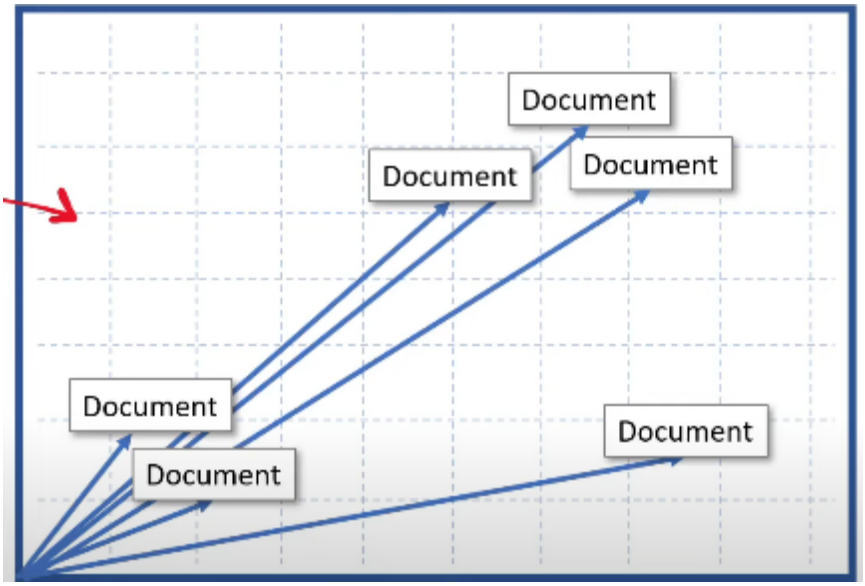
An arrow with a length and a direction.

Vector Space Model

Represents text documents or other types of data as vectors in a high dimensional space.

What are embeddings?

They are vectors of data used by ML models to **find relationships between data**. ML moedls can also create embeddings.



> 💡 Embeddings can be shared across models (multi-model pattern) to help coordinate a task between models.

AI Services

Responsible Use of AI

AI technologies are powerful and have the potential to make great impacts on people's lives. To ensure that such impacts are positive, Microsoft uses the

following principles when it designs and builds AI solutions. You should consider these principles whenever you make use of AI:

- **Fairness.** All AI systems should treat people fairly, regardless of race, belief, gender, sexuality, or other factors.
- **Reliability and safety.** All AI systems should give reliable answers with quantifiable confidence levels.
- **Privacy and security.** All AI systems should secure and protect sensitive data and operate within applicable data protection laws.
- **Inclusiveness.** All AI systems should be available to all users, regardless of their abilities.
- **Transparency.** All AI systems should operate understandably and openly.
- **Accountability.** All AI systems should be run by people who are accountable for the actions of those systems.

Multiple-Service Resource

- You can provision an **AI services** resource that supports multiple different AI services.
- You could create a single resource that enables you to use the **Azure AI Language**, **Azure AI Vision**, **Azure AI Speech**, and other services.

Single-Service Resoures

- Each AI service can be provisioned individually
- This approach enables you to use separate endpoints for each service (for example to provision them in different geographical regions) and to manage access credentials for each service independently.

Endpoints and Keys

To consume the service through the endpoint, applications require the following information:

1. **The endpoint URI**. This is the HTTP address at which the REST interface for the service can be accessed.
2. **A subscription key**. Access to the endpoint is restricted based on a subscription key. When you provision an AI services resource, two keys are created - applications can use either key. You can also regenerate the keys as required to control access to your resource.
3. **The resource location**. When you provision a resource in Azure, you generally assign it to a location, which determines the Azure data center in which the resource is defined.

To interact and integrate Azure AI Services, we use REST APIs and SDK:

- **REST APIs**: Data is submitted in JSON format over an HTTP request (POST, PUT, GET). Then, the result is returned to the client as an HTTP response often with JSON contents
- **SDKs**: It's easier to build more complex solutions by using native libraries for the programming language in which you're developing the application.

AI Services Security and Monitoring

Authentication

- By default, access to Azure AI services resources is restricted by using subscription keys. Management of access to these keys is a primary consideration for security.
- To **regenerate keys**, you can use the Azure portal, or using the `az cognitiveservices account keys regenerate` Azure command-line interface (CLI) command.

```
az cognitiveservices account keys regenerate --name <resourceNa
```

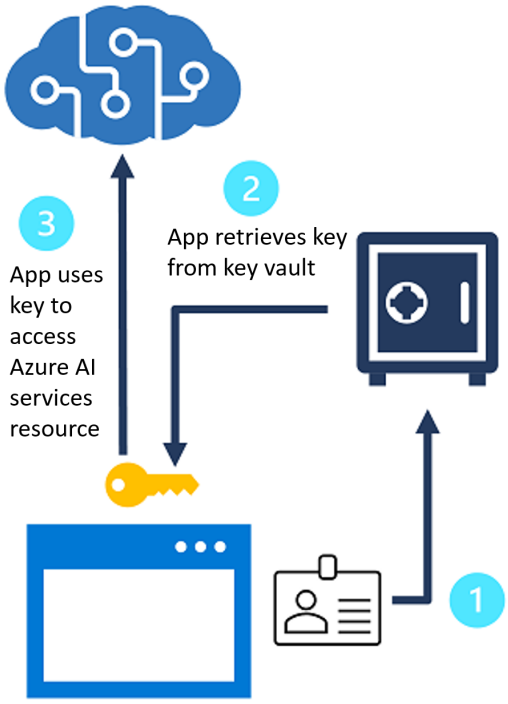
- Each AI service is provided with two keys, enabling you to regenerate keys without service interruption.

To obtain your authentication keys and endpoint, you can use the CLI:

```
az cognitiveservices account keys list --name <resourceName> --
```

Azure Key Vault

- Azure Key Vault is an Azure service in which you can securely store secrets (such as passwords and keys).
- Access to the key vault is granted to *security principals*, which you can think of user identities that are authenticated using Microsoft Entra ID.
- You can store the subscription keys for an AI services resource in Azure Key Vault, and assign a managed identity to client applications that need to use the service. The applications can then retrieve the key as needed from the key vault, without risk of exposing it to unauthorized users.



Create a Service Principal

To access the secret in the key vault, your application must use a service principal that has access to the secret.

```
az ad sp create-for-rbac -n "api://<spName>" --role owner --sc
```

In the previous command, replace *<spName>* with a unique suitable name for an application identity such as 'ai-app'.

> 💡 If you are unsure of your subscription ID, use the ***az account show*** command to retrieve your subscription information - the subscription ID is the **id** attribute in the output. If you see an error about the object already existing, please choose a different unique name.

The output of the previous command for creating a service principal should look like this:

```
{
    "appId": "abcd12345efghi67890jklmn",
    "displayName": "api://ai-app-",
    "password": "1a2b3c4d5e6f7g8h9i0j",
    "tenant": "1234abcd5678fghi90jklm"
}
```

To get the **object ID** of your service principal:

```
az ad sp show --id <appId>
```

To assign permission for your new service principal to access secrets in your Key Vault, run the following command and replace *<objectId>* with the value of your service principal's ID value obtained previously:

```
az keyvault set-policy -n <keyVaultName> --object-id <objectId>
```

💡 In this exercise, we'll store the service principal credentials in the application configuration and use them to authenticate a **ClientSecretCredential** identity in your application code. This is fine for development and testing, but in a real production application, an administrator would assign a *managed identity* to the application so that it uses the service principal identity to access resources, without caching or storing the password.

Monitoring

Azure AI Services can be a critical part of an overall application infrastructure. It's important to be able to monitor activity and get alerted to issues that may need attention.

Alerts

- Microsoft Azure provides alerting support for resources through the creation of *alert rules*.
- You use alert rules to configure notifications and alerts for your resources based on events or metric thresholds
- *Alert rules* are created on the resource you want to monitor.

Metrics

- Azure Monitor collects metrics for Azure resources at regular intervals so that you can track indicators of resource utilization, health, and performance.
- You can view metrics for an individual resource in the Azure portal by selecting the resource and viewing its **Metrics** page.

Manage Diagnostic Logging

Diagnostic logging enables you to capture rich operational data for an Azure AI services resource, which can be used to analyze service usage and troubleshoot problems.

- **Azure Log Analytics:** It is a service that enables you to query and visualize log data within the Azure portal.
- **Azure Storage:** It is a cloud-based data store that you can use to store log archives (which can be exporte for further analysis).

Containers

- A container comprises an application or service and the runtime components needed to run it, while abstracting the underlying operating system and hardware.
- A container is encapsulated in a *container image* that defines the software and configuration it must support. Images can be stored in a central registry, such as *Docker Hub*, or you can maintain a set of images in your own registry.
- To use a container, you typically pull the container image from a registry and deploy it to a container host, specifying any required configuration settings.

The container host can be in the cloud, in a private network, or on your local computer.

Azure AI Services Container

There are container images for Azure AI services in the Microsoft Container Registry that you can use to deploy a containerized service that encapsulates an individual Azure AI services service API.

When you deploy an Azure AI services container image to a host, you must specify three settings.:

- ApiKey ⇒ Key from your deployed Azure AI service; used for billing.
- Billing ⇒ Endpoint URI from your deployed Azure AI service; used for billing.
- EULA ⇒ Value of **accept** to state you accept the license for the container.

To deploy a container image for text translation locally:

```
docker run --rm -it -p 5000:5000 --memory 12g --cpus 1 mcr.micr
```

After you deployed an Azure AI container, it has the following properties:

- **Status**: This should be *Running*.
- **IP Address**: This is the public IP address you can use to access your container instances.
- **FQDN**: This is the *fully-qualified domain name* of the container instances resource, you can use this to access the container instances instead of the IP address.

Azure AI Vision

Azure AI Vision is an artificial intelligence capability that enables software systems to interpret visual input by analyzing images. It provides functionality that you can use for:

- Description and tag generation

- Object detection
- People detection
- Image metadata (width, height, color...), color and type analysis
- Category identification
- Background removal
- Moderation rating
- Optical character recognition (OCR)
- Smart thumbnail generation

> 💡 You can provision **Azure AI Vision** as a single-service resource, or you can use the Azure AI Vision API in a multi-service **Azure AI Services** resource.

To analyze an image, you can use the **Analyze Image** REST method or the equivalent method in the SDK for your preferred programming language. This method returns a JSON document containing the requested information.

```
from azure.ai.vision.imageanalysis import ImageAnalysisClient
from azure.ai.vision.imageanalysis.models import VisualFeatures
from azure.core.credentials import AzureKeyCredential

client = ImageAnalysisClient(
    endpoint=os.environ["ENDPOINT"],
    credential=AzureKeyCredential(os.environ["KEY"])
)

result = client.analyze(
    image_url="<url>",
    visual_features=[VisualFeatures.CAPTION, VisualFeatures.REAI
    gender_neutral_caption=True,
```

```
    language="en",
)
```

Available visual features are contained in the `VisualFeatures` enum:

- 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.DENSE_CAPTIONS: Generates more detailed captions for the objects detected
- VisualFeatures.PEOPLE: Returns the bounding box for detected people
- VisualFeatures.SMART_CROPS: Returns the bounding box of the specified aspect ratio for the area of interest
- VisualFeatures.READ: Extracts readable text

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": [
      {
        "text": "a house in the woods",
        "confidence": 0.7055229544639587,
        "boundingBox": {
          "x": 0,
          "y": 0,
          "w": 640,
          "h": 640
        }
```

```
      },
      ...
  "metadata": {
    "width": 640,
    "height": 640
  }
}
```

Custom Vision Project

To create a custom Azure AI Vision model, you first need an Azure AI Services resource (or an Azure AI Vision resource). Once that resource is deployed to your subscription, you need to create a custom project.

For creating a custom Vision project, you will need:

- **Dataset**: collection of images to use while training your model.
- **COCO file:** Defines the label information about those images.

In most cases, the steps you follow are:

1. Create your **blob storage container** and upload just the training images.
2. Create the dataset for your project, and connect it to your blob storage container. When creating your dataset, you define what type of project it is (image classification, object detection, or product recognition).
3. Label your data in your Azure Machine Learning Data Labeling Project, which creates the COCO file in your blob storage container.
4. Connect your completed COCO file for the labeled images to your dataset.
5. Train your custom model on the dataset and labels created.
6. Verify performance and iterate if the trained performance isn't meeting expectations.

COCO Files

A COCO file is a JSON file with a specific format that defines:

- **images**: Defines the image location in blob storage, name, width, height, and ID.
- **annotations**: Defines the classifications (or objects), including which category the image is classified as, the area, and the bounding box (if labeling for object detection).
- **categories**: Defines the ID for the named label class.

In most cases, COCO files are created by labeling your training images in an Azure Machine Learning Data Labeling Project. If you're migrating from an old Custom Vision project, you can use the migration script to create your COCO file.

> 💡 Once you upload your images to blob storage and created your dataset, the next step is to label your images and connect the resulting COCO file. If you already have a COCO file for your training images, you can skip the labeling step.

> 💡 When the labeling is completed and all training images are correctly classified or labeled, you can add your COCO file to your dataset directly from your Azure Machine Learning workspace.

Face Analysis

There are two Azure AI services that you can use to build solutions that detect faces or people in images.

- **Azure AI Vision service**: Enables you to detect people in an image, as well as returning a bounding box for its location.

- **Face service**: Offers more comprehensive facial analysis capabilities than the Azure AI Vision service, including face detection, comprehensive facial feature analysis, face comparison and verification, and facial recognition.

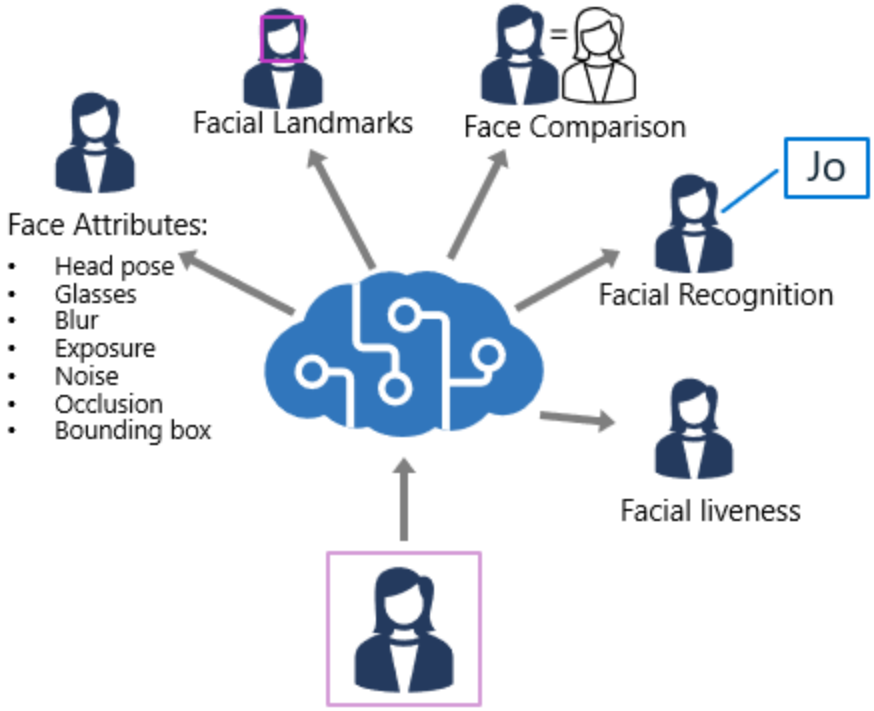
When building a solution that uses facial data, considerations include:

- **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.

To detect and analyze faces with the Azure AI Vision service, call the **Analyze Image** function (SDK or equivalent REST method), specifying **People** as one of the visual features to be returned.

Face Service

The **Face** service provides comprehensive facial detection, analysis, and recognition capabilities.



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 are *similar* or to *verify* that the same person appears in two images.

Reading Text in Images

OCR (Optical Character Recognition) allows you to extract text from images, such as photos of street signs and products, as well as from documents — such as handwritten or unstructured documents.

Image Analysis Optical character recognition (OCR)	**Document Intelligence**
• Use this feature for general, unstructured documents with smaller amount of text, or images that contain text.	• Use this service to read small to large volumes of text from images and PDF documents.
• Results are returned immediately (synchronous) from a single API call.	• This service uses context and structure of the document to improve accuracy.

• Has functionality for analyzing images past extracting text, including object detection, describing or categorizing an image, generating smart-cropped thumbnails and more.	• The initial function call returns an asynchronous operation ID, which must be used in a subsequent call to retrieve the results.
• Examples include: street signs, handwritten notes, and store signs.	• Examples include: receipts, articles, and invoices.

To make an OCR request to **ImageAnalysis**, specify the visual feature as `READ`.

```
result = client.analyze(
    image_url=<image_to_analyze>,
    visual_features=[VisualFeatures.READ]
)
```

If using the REST API, specify the feature as `read`.

```
https://<endpoint>/computervision/imageanalysis:analyze?features
```

The results of the Read OCR function are returned synchronously, either as JSON or the language specific object of a similar structure..

```
{
    "metadata":
    {
        "width": 500,
        "height": 430
    },
    "readResult":
    {
        "blocks":
        [
            {
                "lines":
```

```
                [
                    {
                        "text": "Hello World!",
                        "boundingPolygon":
                        [
                            {"x":251,"y":265},
                            {"x":673,"y":260},
                            {"x":674,"y":308},
                            {"x":252,"y":318}
                        ],
                        "words":
                        [
                            {
                                "text":"Hello",
                                "boundingPolygon":
                                [
                                    {"x":252,"y":267},
                                    {"x":307,"y":265},
                                    {"x":307,"y":318},
                                    {"x":253,"y":318}
                                ],
                            "confidence":0.996
                            },
                            {
                                "text":"World!",
                                "boundingPolygon":
                                [
                                    {"x":318,"y":264},
                                    {"x":386,"y":263},
                                    {"x":387,"y":316},
                                    {"x":319,"y":318}
                                ],
                                "confidence":0.99
                            }
                        ]
                    },
```

```
                ]
            }
        ]
    }
}
```

Azure Video Indexer

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.
- *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.

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>
```

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.

All interactions with the Video Indexer REST API follow the same pattern:

- An initial request to the **AccessToken** method with the API key in the header is used to obtain an access token.
- Subsequent requests use the access token to authenticate when calling REST methods to work with videos.

> 💡 In practice, there are two REST methods: one to get an access token, and another to list the videos in your account.

Azure AI Language

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

- *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.

Capabilities

Language Detection

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).

- **Request:** 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.
- **Response:** 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.

> 💡 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 (*predominant language*).

A certain 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.

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.

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

Extract Linked Entities

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.

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.

Question Answering

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.

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.

	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
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.

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.

When you are 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

```
{
  "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	Score threshold for answers returned.
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 reservatio
      "questions": [
        "How can I cancel a reservation?"
      ],
      "metadata": [
        {
          "name": "category",
          "value": "api"
        }
      ]
    }
  ]
}
```


Improve a Knowledge Base

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

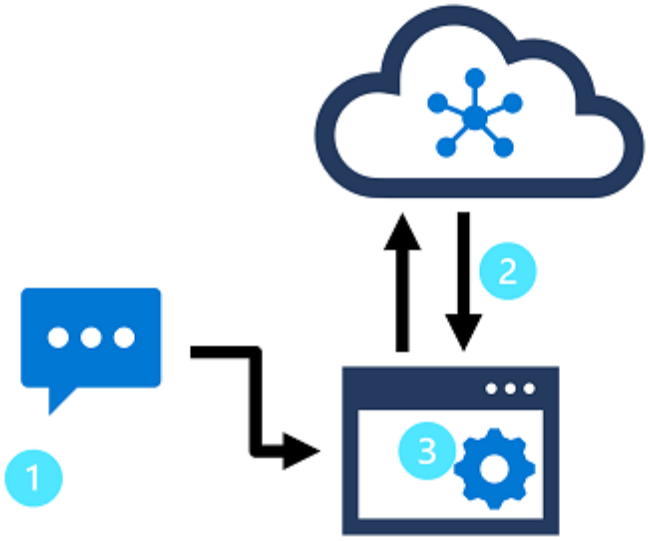
- Active Learning: Active learning can help you make continuous improvements to get better at answering user questions correctly over time. People often ask questions that are phrased differently, but ultimately have the same meaning. Active learning is enabled by default.
- Synonyms: Synonyms are useful when questions submitted by users might include multiple different words to mean the same thing.


```
{
    "synonyms": [
        {
            "alterations": [
```

```
                "reservation",
                "booking"
                ]
        }
    ]
}
```

Natural Language Understanding

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.



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 service features fall into two categories: Pre-configured features, and Learned features.

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.

- **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.
- **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.
- **Key phrase extraction:** Key phrase extraction is a feature that quickly pulls the main concepts out of the provided text.
- **Sentiment analysis:** Sentiment analysis identifies how positive or negative a string or document is.
- **Language detection:** Language detection takes one or more documents, and identifies the language for each.

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.

- **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.

- **Custom named entity recognition:** Custom entity recognition takes custom labeled data and extracts specified entities from unstructured text.
- **Custom text classification**: Custom text classification enables users to classify text or documents as custom defined groups.
- **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.

💡 ***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.

- **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?
- **None**:
  - "Hello"
  - "Goodbye"

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.

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.

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"

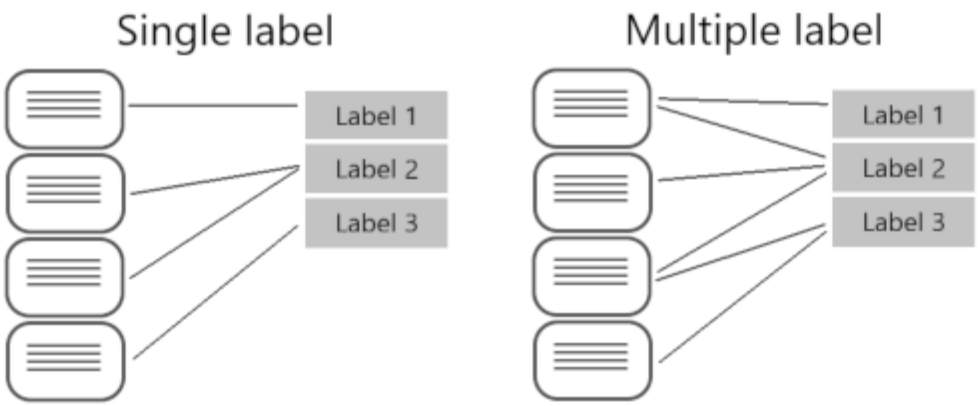
Creating a model is an iterative process with the following activities:

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

Text Classification

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".



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.

Model Evaluation

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

API Payload

To submit a classification task, the API requires the JSON body to specify which task to execute.

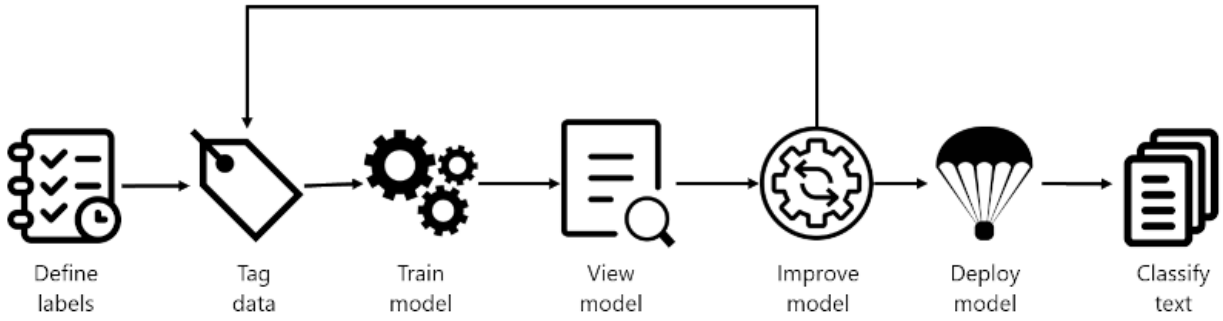
- Single label classification models specify a project type of `customSingleLabelClassification`
- Multiple label classification models specify a project type of `CustomMultiLabelClassification`

Azure AI Language Project Life Cycle

- **Define labels**: Understanding the data you want to classify, identify the possible labels you want to categorize into.
- **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.

- **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.
- **Deploy model**: Once your model performs as desired, deploy your model to make it available via the API.
- **Classify text**: Use your model for classifying text.



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. (80% of your labeled data)
- **Testing** - The testing dataset is labeled data used to verify you 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** - 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** - 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.

Example of using API:

```
# To submit something
curl - X POST <YOUR-ENDPOINT>/language/analyze-text/projects/<PR

# To get something
curl -X GET <ENDPOINT>/language/analyze-text/jobs/<JOB-ID>?api-v
```

Custom Named Entity Recognition (NER)

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.

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`.

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-V
ERSION>
```

Examples of when to use the built-in NER include finding locations, names, or URLs in long text documents.

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.

Considerations for Data Selection and Refining Entities

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.
- **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.

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.

Labeling Data

- **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.

Metrics

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.
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
F1 score	Combination of precision and recall providing a single scoring metric

- 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.

The confusion matrix allows you to visually identify where to add data to improve your model's performance.

Azure AI Translator

Azure AI Translator provides a multilingual text translation API that you can use for:

- Language detection.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/dete
```

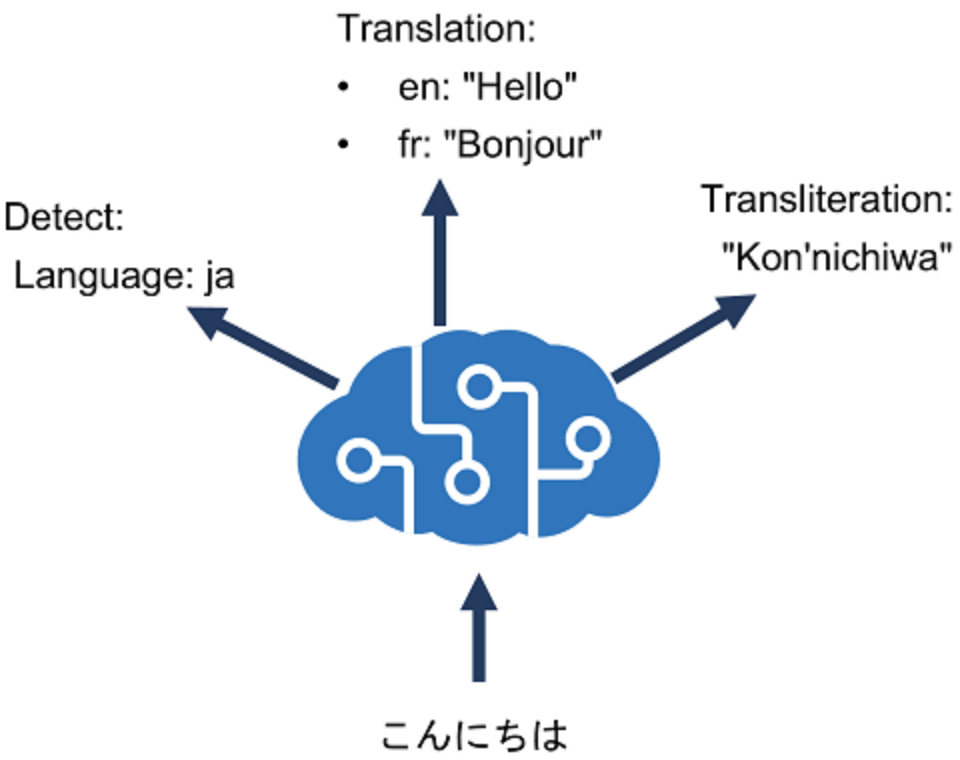
- One-to-many translation.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/tra
```

- Script transliteration (converting text from its native script to an alternative script).

```
curl -X POST "https://api.cognitive.microsofttranslator.com/tra
```

The Azure AI Translator provides an API for translating text between 135 supported languages.



> 💡 You can provision a single-service Azure AI Translator resource, or you can use the Text Analytics API in a multi-service Azure AI Services resource.

Translation Options

Word Alignment

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.

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`.

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.

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 in that have specific vocabularies of terms that require custom translation.

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.

To initiate a translation, you send a **POST** request to the following request URL:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/tra
from=en&to=nl&category=<category-id>" -H "Ocp-Apim-Subscription
```

```
 -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text'
```

The response returns a response code of `200` if the request was successful. 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).

Azure AI Speech

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.
- **Speaker Recognition**: An API that enables your application to recognize individual speakers based on their voice.
- **Intent Recognition**: An API that uses conversational language understanding to determine the semantic meaning of spoken input.

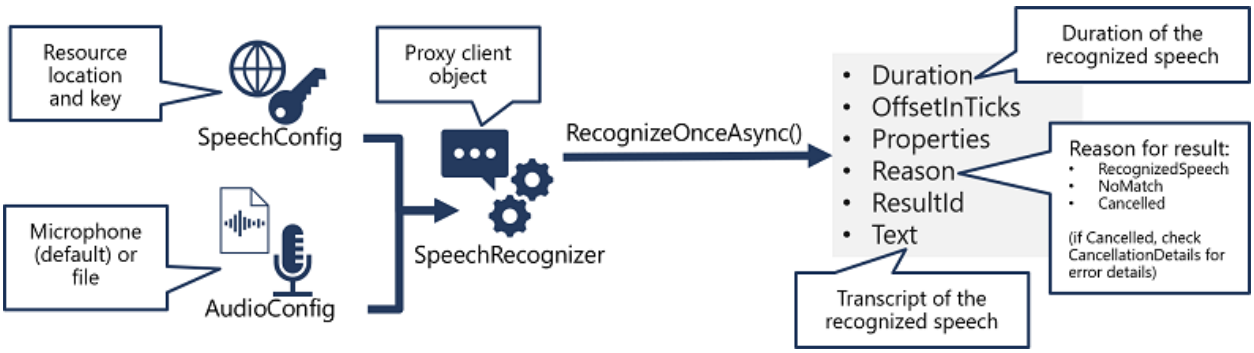
After you create your resource, you'll need the following information to use it from a client application through one of the supported SDKs:

- The *location* in which the resource is deployed (for example, *eastus*)
- One of the *keys* assigned to your resource.

The Azure AI Speech service supports speech recognition through two REST APIs:

- The **Speech to text** API, which is the primary way to perform speech recognition.
- The **Speech to text Short Audio** API, which is optimized for short streams of audio (up to 60 seconds).

Speech to Text API



1. Use a **SpeechConfig** object to encapsulate the information required to connect to your Azure AI Speech resource. Specifically, its **location** and **key**.
2. 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.
3. Use the **SpeechConfig** and **AudioConfig** to create a **SpeechRecognizer** object. This object is a proxy client for the **Speech to text** API.
4. 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.
5. 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

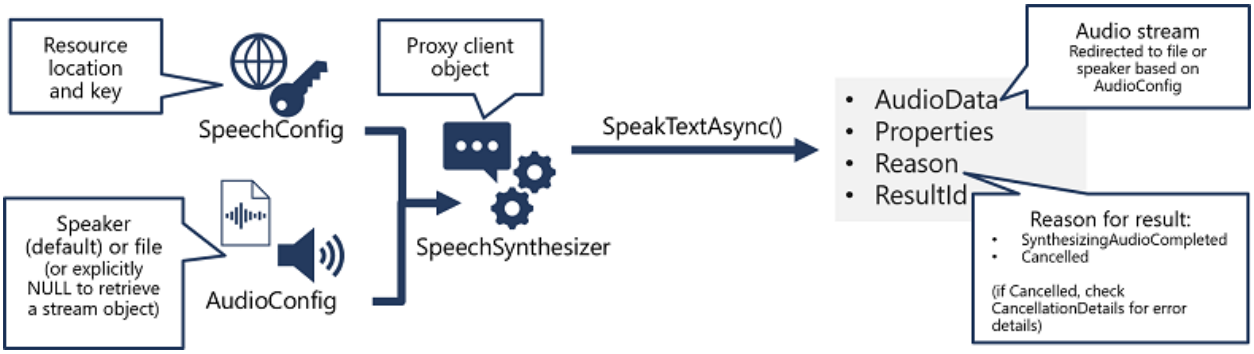
- Reason
- ResultId
- Text

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).

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.



1. Use a **SpeechConfig** object to encapsulate the information required to connect to your Azure AI Speech resource. Specifically, its **location** and **key**.
2. 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.

3. Use the **SpeechConfig** and **AudioConfig** to create a **SpeechSynthesizer** object. This object is a proxy client for the **Text to speech** API.
4. 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.
5. 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).

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

To specify the required output format, use the **SetSpeechSynthesisOutputFormat** method of the **SpeechConfig** object:

```
speechConfig.SetSpeechSynthesisOutputFormat(SpeechSynthesisOutpu
```

Voices

The Azure AI Speech service provides multiple voices that you can use to personalize your speech-enabled applications. There are two kinds of voice that you can use:

- *Standard voices* - synthetic voices created from audio samples.
- *Neural voices* - more natural sounding voices created using deep neural networks.

Set its **SpeechSynthesisVoiceName** property to the voice you want to use:

```
speechConfig.SpeechSynthesisVoiceName = "en-GB-George";
```

Speech Synthesis Markup Language (SSML)

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.

Examples of SSML:

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
                     xmlns:mstts="https://www.w3.org/2001/mstts"
    <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"> toma
        <break strength="weak"/>Lets call the whole thing off!
    </voice>
</speak>
```

This SSML specifies a spoken dialog between two different neural voices, like this:

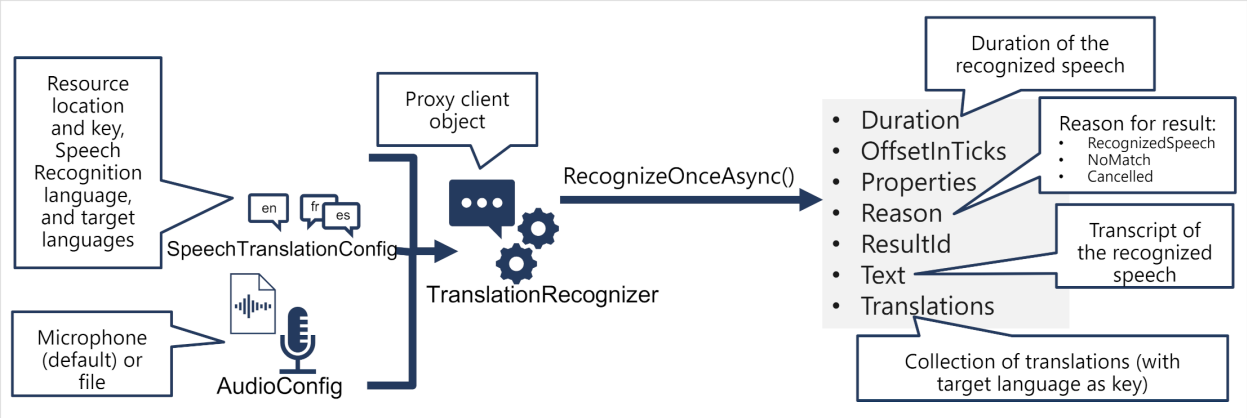
- **Ariana** (*cheerfully*): "I say tomato:
- **Guy**: "I say tomato (pronounced *tom-ah-toe*) ... Let's call the whole thing off!"

To submit an SSML description to the Speech service, you can use the **SpeakSsmlAsync()** method, like this:

```
speechSynthesizer.SpeakSsmlAsync(ssml_string);
```

Translate Speech

Translate Speech to Text



1. Use a **SpeechTranslationConfig** object to encapsulate the information required to connect to your Azure AI Speech resource. Specifically, its location and key.
2. 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.
3. 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.
4. Use the **SpeechTranslationConfig**, and **AudioConfig** to create a **TranslationRecognizer** object. This object is a proxy client for the Azure AI Speech translation API.
5. 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.
6. Process the response from Azure AI Speech. In the case of the **RecognizeOnceAsync**() method, the result is a **SpeechRecognitionResult** object.

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.

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.

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:

1. Use a **TranslationRecognizer** to translate spoken input into text transcriptions in one or more target languages.
2. Iterate through the **Translations** dictionary in the result of the translation operation, using a **SpeechSynthesizer** to synthesize an audio stream for each language.

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. With Azure AI Search, you can:

- Index documents and data from a range of sources.
- Use cognitive skills to enrich index data.
- Store extracted insights in a knowledge store for analysis and integration.

Management and Capacity

Service Tiers

The available pricing tiers are:

- **Free (F)**. Use this tier to explore the service or try the tutorials in the product documentation.
- **Basic (B)**: Use this tier for small-scale search solutions that include a maximum of 15 indexes and 2 GB of index data.
- **Standard (S)**: Use this tier for enterprise-scale solutions. There are multiple variants of this tier, including **S**, **S2**, and **S3**; which offer increasing capacity in terms of indexes and storage, and **S3HD**, which is optimized for fast read performance on smaller numbers of indexes.
- **Storage Optimized (L)**: Use a storage optimized tier (**L1** or **L2**) when you need to create large indexes, at the cost of higher query latency.

Replicas and Partitions

Depending on the pricing tier you select, you can optimize your solution for scalability and availability by creating *replicas* and *partitions*.

- *Replicas* are instances of the search service - you can think of them as nodes in a cluster. Increasing the number of replicas can help ensure there is sufficient capacity to service multiple concurrent query requests while managing ongoing indexing operations.

- *Partitions* are used to divide an index into multiple storage locations, enabling you to split I/O operations such as querying or rebuilding an index.

The combination of replicas and partitions you configure determines the *search units* used by your solution. Put simply, the number of search units is the number of replicas multiplied by the number of partitions (R x P = SU). For example, a resource with four replicas and three partitions is using 12 search units.

Search Components

An AI Search solution consists of multiple components, each playing an important part in the process of extracting, enriching, indexing, and searching data.

Data Source

Azure AI Search supports multiple types of data source, including:

- Unstructured files in Azure blob storage containers.
- Tables in Azure SQL Database.
- Documents in Cosmos DB.

Azure AI Search can pull data from these data sources for indexing.

Alternatively, applications can push JSON data directly into an index, without pulling it from an existing data store.

Skillset

The skills used by an indexer are encapsulated in a *skillset* that defines an enrichment pipeline in which each step enhances the source data with insights obtained by a specific AI skill. Examples of the kind of information that can be extracted by an AI skill include:

- The language in which a document is written.
- Key phrases that might help determine the main themes or topics discussed in a document.
- A sentiment score that quantifies how positive or negative a document is.
- Specific locations, people, organizations, or landmarks mentioned in the content.

- AI-generated descriptions of images, or image text extracted by optical character recognition.
- Custom skills that you develop to meet specific requirements.

Indexer

The *indexer* is the engine that drives the overall indexing process. It takes the outputs extracted using the skills in the skillset, along with the data and metadata values extracted from the original data source, and maps them to fields in the index.

An indexer is automatically run when it is created, and can be scheduled to run at regular intervals or run on demand to add more documents to the index.

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.

Indexing Process

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 by cognitive 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**

You can configure the indexer to extract the image data and place each image in a **normalized_images** collection. Furthermore, a skill that detects the *language* in which a document is written might store its output in a **language** field.

- **document**
  - **metadata_storage_name**
  - **metadata_author**
  - **content**
  - **normalized_images**
    - **image0**
      - Text
    - **image1**
      - Text
  - Language

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.

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.

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:

1. Fields extracted directly from the source data are all mapped to index fields. These mappings can be *implicit* (fields are automatically mapped to in 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).
2. 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.

Searching an Index

After you have created and populated an index, you can query it to search for information in the indexed document content. 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

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.

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

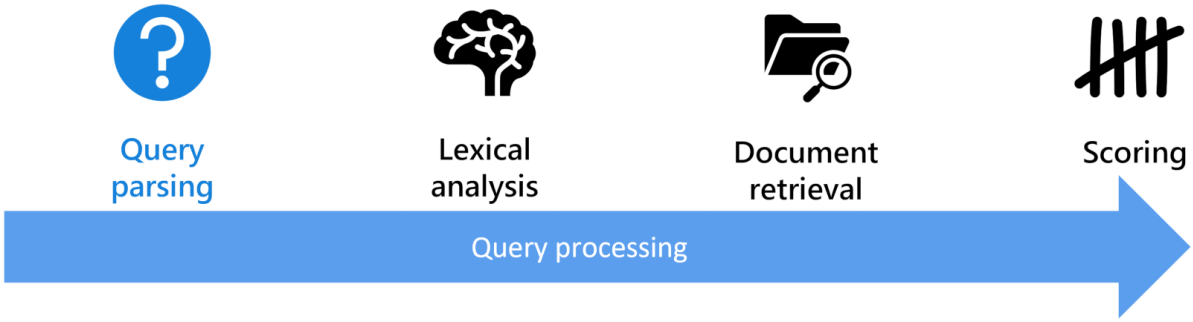
Client applications submit queries to Azure AI Search by specifying a search expression along with other parameters that determine how the expression is evaluated and the results returned.

- **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.

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 (nonessential *stopwords* are removed, words converted to root form, composite words are split, and so on).

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.



Filtering Results

You can apply filters to queries in two ways.

For example, suppose you want to find documents containing the text *London* that have an **author** field value of *Reviewer*.

- By including filter criteria in a `simple` **search** expression.

```
search=London+author='Reviewer'
queryType=Simple
```

- By providing an OData filter expression as a **$filter** parameter with a `full` `syntax` search expression. In the following expression, it was used `$filter` with a *full* Lucene search expression.

```
search=London
$filter=author eq 'Reviewer'
queryType=Full
```

💡 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.

```
// Return all of the possible values for 'author'
search=*
facet=author
```

```
// Subsequent query that filters the previous 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*).

```
// Get the first elements listed
search=*
$orderby=last_modified desc
```

Enhancing the Index

Search-As-You-Type

By adding a *suggester* to an index, you can enable two forms of search-as-you-type experience to help users find relevant results more easily:

- *Suggestions* - retrieve and display a list of suggested results as the user types into the search box, without needing to submit the search query.
- *Autocomplete* - complete partially typed search terms based on values in index fields.

To implement one or both of these capabilities, create or update an index, defining a suggester for one or more fields.

Custom Scoring and Result Boosting

By default, search results are sorted by a relevance score that is calculated based on a term-frequency/inverse-document-frequency (TF/IDF) algorithm. You can customize the way this score is calculated by defining a *scoring profile* that applies a weighting value to specific fields - essentially increasing the search score for documents when the search term is found in those fields.

After you've defined a scoring profile, you can specify its use in an individual search, or you can modify an index definition so that it uses your custom scoring profile by default.

Synonyms

To help users find the information they need, you can define *synonym maps* that link related terms together. You can then apply those synonym maps to individual fields in an index, so that when a user searches for a particular term, documents with fields that contain the term or any of its synonyms will be included in the results.

Custom Skill for AI Search

You can implement custom skills as web-hosted services (such as Azure Functions) that support the required interface for integration into a skillset.

Your custom skill must implement the expected schema for input and output data that is expected by skills in an Azure AI Search skillset.

- **Input Schema:** The input schema for a custom skill defines a JSON structure containing a record for each document to be processed. Each document has a unique identifier, and a data payload with one or more inputs.

- **Output Schema:** The schema for the results returned by your custom skill reflects the input schema. It is assumed that the output contains a record for each input record, with either the results produced by the skill or details of any errors that occurred.

```
{
    "values": [
      {
        "recordId": "<unique_identifier_from_input>",
        "data":
           {
             "<output1_name>":  "<output1_value>",
              ...
           },
         "errors": [...],
         "warnings": [...]
      },
      {
        "recordId": "< unique_identifier_from_input>",
        "data":
           {
             "<output1_name>":  "<output1_value>",
              ...
           },
         "errors": [...],
         "warnings": [...]
      },
      ...
    ]
}
```

To integrate a custom skill into your indexing solution, you must add a skill for it to a skillset using the `Custom.WebApiSkill` skill type.

The skill definition must:

- Specify the URI to your web API endpoint, including parameters and headers if necessary.
- Set the context to specify at which point in the document hierarchy the skill should be called
- Assign input values, usually from existing document fields
- Store output in a new field, optionally specifying a target field name (otherwise the output name is used)

Knowledge Store

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.

Projections

The projections of data to be stored in your knowledge store are based on the document structures generated by the enrichment pipeline in your indexing process. Each skill in your skillset iteratively builds a JSON representation of the enriched data for the documents being indexed, and you can persist some or all of the fields in the document as projections.

- To simplify (reduce complexity of) the mapping of these field values to projections in a knowledge store, it's common to use the **Shaper** skill to create a new, field containing a simpler structure for the fields you want to map to projections.

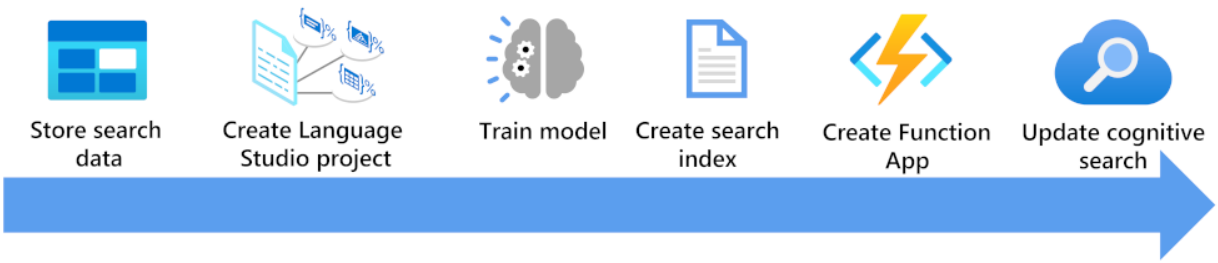
Define a Knowledge Store

To define the knowledge store and the projections you want to create in it, you must create a `knowledgeStore` object in the skillset that specifies the Azure Storage connection string for the storage account where you want to create projections, and the definitions of the projections themselves.

You can define **object** projections (for indexing documents), **table** projections (for a relational schema purpose), and **file** projections (indexing images) depending on what you want to store.

Enrich Data with Azure AI Language

Here, you'll see what you need to consider to enrich a search index using a custom text classification model.



- Store your documents so they can be accessed by Language Studio and Azure AI Search indexers
- Create a custom text classification project
- Train and test your model
- Create a search index based on your stored documents
- Create a function app that uses your deployed trained model
- Update your search solution, your index, indexer, and custom skillset

> 💡 Azure Blob storage can be accessed from both Language Studio and Azure AI Services. The container needs to be accessible, so the simplest option is to choose Container, but it's also possible to use private containers with some additional configuration.

When creating a function app you will need to know these five things:

1. The text to be classified.
2. The endpoint for your trained custom text classification deployed model.
3. The primary key for the custom text classification project.

4. The project name.
5. The deployment name.

For updating the AI Search solution, there are three changes in the Azure portal you need to make to enrich your search index.

1. You need to add a field to your index to store the custom text classification enrichment.
2. You need to add a custom skillset to call your function app with the text to classify.
3. You need to map the response from the skillset into the index.

Advanced Search Features

Improve Search Query with Term Boosting

Use Lucene query parser. First, you have to enable it by adding `&queryType=full` to the query string.

An example of this Lucene query is like this:

```
search=(Description:luxury OR Category:luxury^3) AND Tags:'air con'*&$select=HotelId, HotelName, Category, Tags, Description&$count=true&queryType=full
```

Improve Relevance of Results by Adding Scoring Profiles

Azure AI Search uses the BM25 similarity ranking algorithm. The algorithm scores documents based on the search terms used.

The score is a function of the number of times identified search terms appear in a document, the document's size, and the rarity of each of the terms. By default, the search results are ordered by their search score, highest first.

> 💡 If two documents have an identical search score, you can break the tie by adding an `$orderby` clause.

The most straightforward scoring profile defines different weights for fields in an index.

The scoring profile can also include functions, for example, distance or freshness. Functions provide more control than simple weighting, for example, you can define the boosting duration applied to newer documents before they score the same as older documents.

The power of scoring profiles means that instead of boosting a specific term in a search request, you can apply a scoring profile to an index so that fields are boosted automatically for all queries.

The functions available to add to a scoring profile are:

Function	Description
Magnitude	Alter scores based on a range of values for a numeric field
Freshness	Alter scores based on the freshness of documents as given by a DateTimeOffset field
Distance	Alter scores based on the distance between a reference location and a GeographyPoint field
Tag	Alter scores based on common tag values in documents and queries

Improve an Index with Analyzers and Tokenized Terms

In AI Search, this kind of processing is performed by analyzers. If you don't specify an analyzer for a field, the default Lucene analyzer is used. The default Lucene analyzer is a good choice for most fields because it can process many languages and return useful tokens for your index.

Alternatively, you can specify one of the analyzers that are built into AI Search. Built-in analyzers are of two types:

- **Language analyzers.** If you need advanced capabilities for specific languages, such as lemmatization, word decompounding, and entity recognition, use a built-in language analyzer. Microsoft provides 50 analyzers for different languages.

- **Specialized analyzers.** These analyzers are language-agnostic and used for specialized fields such as zip codes or product IDs. You can, for example, use the **PatternAnalyzer** and specify a regular expression to match token separators.

A custom analyzer (with unusual behavior for a field) consists of:

- **Character filters**. These filters process a string before it reaches the tokenizer.
  - **html_strip.** This filter removes HTML constructs such as tags and attributes.
  - **mapping.** This filter enables you to specify mappings that replace one string with another. For example, you could specify a mapping that replaces *TX* with *Texas*.
  - **pattern_replace.** This filter enables you to specify a regular expression that identifies patterns in the input text and how matching text should be replaced.
- **Tokenizers**. These components divide the text into tokens to be added to the index.
  - **classic.** This tokenizer processes text based on grammar for European languages.
  - **keyword.** This tokenizer emits the entire input as a single token. Use this tokenizer for fields that should always be indexed as one value.
  - **lowercase.** This tokenizer divides text at non-letters and then modifies the resulting tokens to all lower case.
  - **microsoft_language_tokenizer.** This tokenizer divides text based on the grammar of the language you specify.
  - **pattern.** This tokenizer divides texts where it matches a regular expression that you specify.
  - **whitespace.** This tokenizer divides text wherever there's white space.

- **Token filters.** These filters remove or modify the tokens emitted by the tokenizer.
    - Language-specific filters, such as **arabic_normalization**. These filters apply language-specific grammar rules to ensure that forms of words are removed and replaced with roots.
    - **apostrophe**. This filter removes any apostrophe from a token and any characters after the apostrophe.
    - **classic.** This filter removes English possessives and dots from acronyms.
    - **keep.** This filter removes any token that doesn't include one or more words from a list you specify.
    - **length.** This filter removes any token that is longer than your specified minimum or shorter than your specified maximum.
    - **trim.** This filter removes any leading and trailing white space from tokens.

Enhance an Index to Include Multiple Languages

Support for multiple languages can be added to a search index. You can add language support manually by providing all the translated text fields in all the different languages you want to support.

To add multiple languages to an index, first, identify all the fields that need a translation. Then duplicate those fields for each language you want to support.

For example, if an index has an English description field, you'd add `description_fr` for the French translation and `description_de` for German.

Ordering Results by Distance from a Given Reference Point

To help you compare locations on the Earth's surface, AI Search includes geo-spatial functions that you can call in queries.

To ask AI Search to return results based on their location information, you can use two functions in your query:

- `geo.distance`. This function returns the distance (in km.) in a straight line across the Earth's surface from the point you specify to the location of the search

result.

```
search=(Description:luxury OR Category:luxury)$filter=geo.distance(location, geography'POINT(-122.131577 47.678581)') le 5&$select=HotelId, HotelName, Category, Tags, Description&$count=true
```

- `geo.intersects`. This function returns `true` if the location of a search result is inside a polygon that you specify. The following points represent a square (first coordinates are equal to the last ones ⇒ closed polygon).

```
search=(Description:luxury OR Category:luxury) AND geo.intersects(Location, geography'POLYGON((2.32 48.91, 2.27 48.91, 2.27 48.60, 2.32 48.60, 2.32 48.91))')&$select=HotelId, HotelName, Category, Tags, Description&$count=true
```

To use these functions, make sure that your index includes the location for results. Location fields should have the datatype `Edm.GeographyPoint` and store the latitude and longitude.

Custom Azure Machine Learning Skillset

When you enrich a search index with an Azure Machine Learning (AML) custom skill, the enrichment happens at the document level. The skillset used by your document indexer needs to include an `AmlSkill`.

```
{
      "@odata.type": "#Microsoft.Skills.Custom.AmlSkill",
      "name": "AML name",
      "description": "AML description",
      "context": "/document",
      "uri": "https://[Your AML endpoint]",
      "key": "Your AML endpoint key",
      "resourceId": null,
      "region": null,
      "timeout": "PT30S",
      "degreeOfParallelism": 1,
      "inputs": [
```

```
        ...
        ]
}
```

The remaining settings that control the performance of the skill are `timeout` and `degreeOfParallelism`.

- *Create an AML workspace*: When you create the AML workspace, Azure will also create storage accounts, a key store, and application insights resources.
- *Create a train model*
- *Edit scoring code*
- *Create endpoint*: The model is deployed to an endpoint. Azure AI Machine Learning Studio supports deploying a model to a real-time endpoint, a batch endpoint, or a web service. The other restriction is that the endpoint has to be an Azure Kubernetes Service (AKS), container instances aren't supported.
- *Update cognitive search*

Azure Data Factory

ADF comes with connections to nearly 100 different data stores. With connectors like HTTP and REST that allow you to connect an unlimited number of data stores. These data stores are used as a source or a target (called sinks in the copy activity) in pipelines.

The steps you need to take to use and ADF pipeline to push data into a search index are:

1. Create an Azure AI Search index with all the fields you want to store data in.
2. Create a pipeline with a copy data step.
3. Create a data source connection to where your data resides.
4. Create a sink to connect to your search index.
5. Map the fields from your source data to your search index.
6. Run the pipeline to push the data into the index.

At the moment, the Azure AI Search linked service as a sink only supports these fields:

- String
- Int32
- Int64
- Double
- Boolean
- DateTImeOffset

Push Data Outside Using REST API

The REST API is the most flexible way to push data into an Azure AI Search index.

There are two supported REST APIs provided by AI Search. Search and management APIs. This module focuses on the search REST APIs that provide operations on five features of search:

Feature	Operations
Index	Create, delete, update, and configure.
Document	Get, add, update, and delete.
Indexer	Configure data sources and scheduling on limited data sources.
Skillset	Get, create, delete, list, and update.
Synonym map	Get, create, delete, list, and update.

If you want to call any of the search APIs you need to:

- Use the HTTPS endpoint (over the default port 443) provided by your search service, you must include an **api-version** in the URI.
- The request header must include an **api-key** attribute.

The JSON in the HTTP POST must be in this format:

```
{
  "value": [
    {
      "@search.action": "upload (default) | merge | mergeOrUp
load | delete",
      "key_field_name": "unique_key_of_document", (key/value
pair for key field from index schema)
      "field_name": field_value (key/value pairs matching ind
ex schema)
        ...
    },
    ...
  ]
}
```

You can add as many documents in the value array as you want. However, for optimal performance consider batching the documents in your requests up to a maximum of 1,000 documents, or 16 MB in total size.

Use .NET Core to Index Any Data

For best performance use the latest `Azure.Search.Document` client library, currently version 11. You can install the client library with NuGet:

```
dotnet add package Azure.Search.Documents --version 11.4.0
```

How your index performs is based on six key factors:

- The search service tier and how many replicas and partitions you've enabled.
- The complexity of the index schema. Reduce how many properties (searchable, facetable, sortable) each field has.
- The number of documents in each batch, the best size will depend on the index schema and the size of documents.
- How multithreaded your approach is.
- Handling errors and throttling. Use an exponential backoff retry strategy.

> 💡 If your index starts to throttle requests due to overloads, it responds with a 503 (request rejected due to heavy load) or 207 (some documents failed in the batch) status. You have to handle these responses and a good strategy is to backoff. Backing off means pausing for some time before retrying your request again. If you increase this time for each error, you'll be exponentially backing off.
>
> A function `ExponentialBackoffAsync` that implements the backoff strategy.

- Where your data resides, try to index your data as close to your search index. For example, run uploads from inside the Azure environment.

Managing Security

When you think about securing your search solution, you can focus on three areas:

- Inbound search requests made by users to your search solution
- Outbound requests from your search solution to other servers to index documents
- Restricting access at the document level per user search request

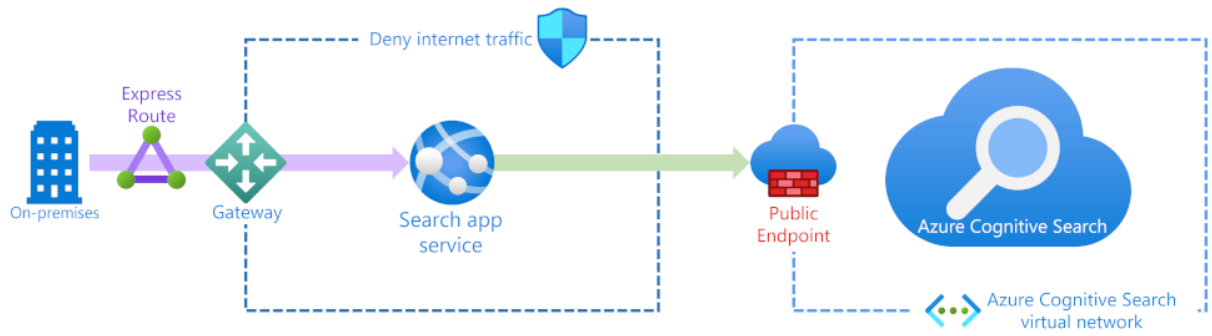
Data Encryption

The Azure AI Search service, like all Azure services, encrypts the data it stores at rest with service-managed keys. This encryption includes indexes, data sources, synonym maps, skillsets, and even the indexer definitions.

Data in transit is encrypted using the standard HTTPS TLS 1.3 encryption over port 443.

Secure Inbound Traffic

If your search service is only going to be used by on-premises resources, you can harden security with an ExpressRoute circuit, Azure Gateway, and an App service. There's also the option to change the public endpoint to use an Azure private link. You'll also need to set up an Azure virtual network and other resources. Using a private endpoint is the most secure solution, although it does come with the added cost of using those services that need to be hosted on the Azure platform.



The default option when you create your ACS is key-based authentication. There are two different kinds of keys:

- **Admin keys** - grant your write permissions and the right to query system information (*maximum of 2 admin keys can be created per search service*)
- **Query keys** - grant read permissions and are used by your users or apps to query indexes (*maximum of 50 query keys can be created per search service*)

Role-based access control (RBAC) is provided by the Azure platform as a global system to control access to resources. You can use RBAC in Azure AI Search in the following ways:

- Roles can be granted access to administer the service

- Define roles with access to create, load, and query indexes

The built-in roles you can assign to manage the Azure AI Search service are:

- **Owner** - Full access to all search resources
- **Contributor** - Same as above, but without the ability to assign roles or change authorizations
- **Reader** - View partial service information

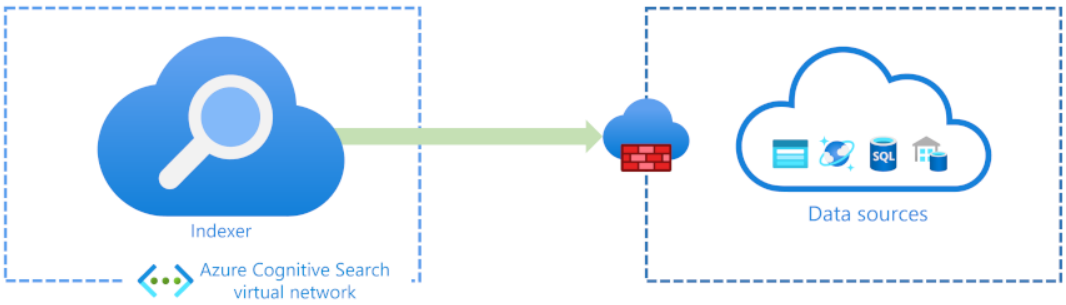
If you need a role that can also manage the data plane for example search indexes or data sources, use one of these roles:

- **Search Service Contributor** - A role for your search service administrators (the same access as the Contributor role above) and the content (indexes, indexers, data sources, and skillsets)
- **Search Index Data Contributor** - A role for developers or index owners who will import, refresh, or query the documents collection of an index
- **Search Index Data Reader** - Read-only access role for apps and users who only need to run queries

Secure Outbound Traffic

If your data sources are hosted on the Azure platform, you can also secure connections using a system or user-assigned managed identity.

Azure services can restrict access to them using a firewall. Your firewall can be configured to only allow the IP address of your Azure AI Search service. If you're enriching your indexes with AI, you'll also need to allow all the IP addresses in the **AzureCognitiveSearch** service tag.



Secure Data at the Document-Level

Controlling who has access at the document level requires you to update each document in your search index. You need to add a new security field to every document that contains the user or group IDs that can access it. The security field needs to be filterable so that you can filter search results on the field.

With this field in place and populated with the allowed user or groups, you can restrict results by adding the `search.in` filter to all your search queries. If you're using HTTP POST requests, the body should look like this:

```
{
   "filter":"security_field/any(g:search.in(g, 'user_id1, group_
}
```

Optimize Performance of Search Solutions

You can't optimize when you don't know how well your search service performs. Create a baseline performance benchmark so you can validate the improvements you make, but you can also check for any degradation in performance over time using Log Analytics.

Checking Throttles

Azure AI Search searches and indexes can be throttled.

- If your users or apps are having their searches throttled, it's captured in Log Analytics with a 503 HTTP responses.
- If your indexes are being throttled, they'll show up as 207 HTTP responses.

This query you can run against your search service logs shows you if your search service is being throttled (it shows HTTPS responses):

```
AzureDiagnostics
| where TimeGenerated > ago(7d)
| summarize count() by resultSignature_d
| render barchart
```

Check Performance of Individual Queries

The best way to test individual query performance is with a client tool like Postman. You can use any tool that will show you the headers in the response to a query. Azure AI Search will always return an 'elapsed-time' value for how long it took the service to complete the query.

To calculate how long it would take to send and then receive the response from the client, subtract the elapsed time from the total round trip.

Optimize your Index Size and Schema

How your search queries perform is directly connected to the size and complexity of your indexes. The smaller and more optimized your indexes, the fast Azure AI Search can respond to queries.

Improve the Performance of your Queries

Use this checklist for writing better queries:

1. Only specify the fields you need to search using the **searchFields** parameter. As more fields require extra processing.
2. Return the smallest number of fields you need to render on your search results page. Returning more data takes more time.
3. Try to avoid partial search terms like prefix search or regular expressions. These kinds of searches are more computationally expensive.
4. Avoid using high skip values. This forces the search engine to retrieve and rank larger volumes of data.
5. Limit using facetable and filterable fields to low cardinality data.
6. Use search functions instead of individual values in filter criteria. For example, you can use `search.in(userid, '123,143,563,121',',')` instead of `$filter=userid eq 123 or userid eq 143 or userid eq 563 or userid eq 121`.

💡 Choosing the best tier for your search solution requires you to know the approximate total size of storage you're going to need. The largest index supported currently is 12 partitions in the L2 tier offering a total of 24 TB.

Manage Costs

Using a given information an estimate for an S2 tier search solution, using four search units (SU), extracting 80,000 images, and using 200,000 semantic queries would be:

Item	Estimate
S2 tier 4SU	$981.12 * 4 = **$3,924.48**
Cracking images	1$ * 80 = **$80**
Semantic search	**$500**
Total estimate	**$4,504.48** per month

The final costs related to running a search service are the data ingestion and storage costs. So the above estimate doesn't include other infrastructure costs you can accrue. These other costs would be things like the storage and processing of your source data.

These tips can help you reduce the cost of running your search solution:

1. Minimize bandwidth costs by using as few regions as possible. Ideally, all the resources should reside in the same region.
2. If you have predictable patterns of indexing new data, consider scaling up inside your search tier. Then scale back down for your regular querying.
3. To keep your search requests and responses inside the Azure datacenter boundary, use an Azure Web App front-end as your search app.
4. Enable enrichment caching if you're using AI enrichment on blob storage.

Improve Reliability

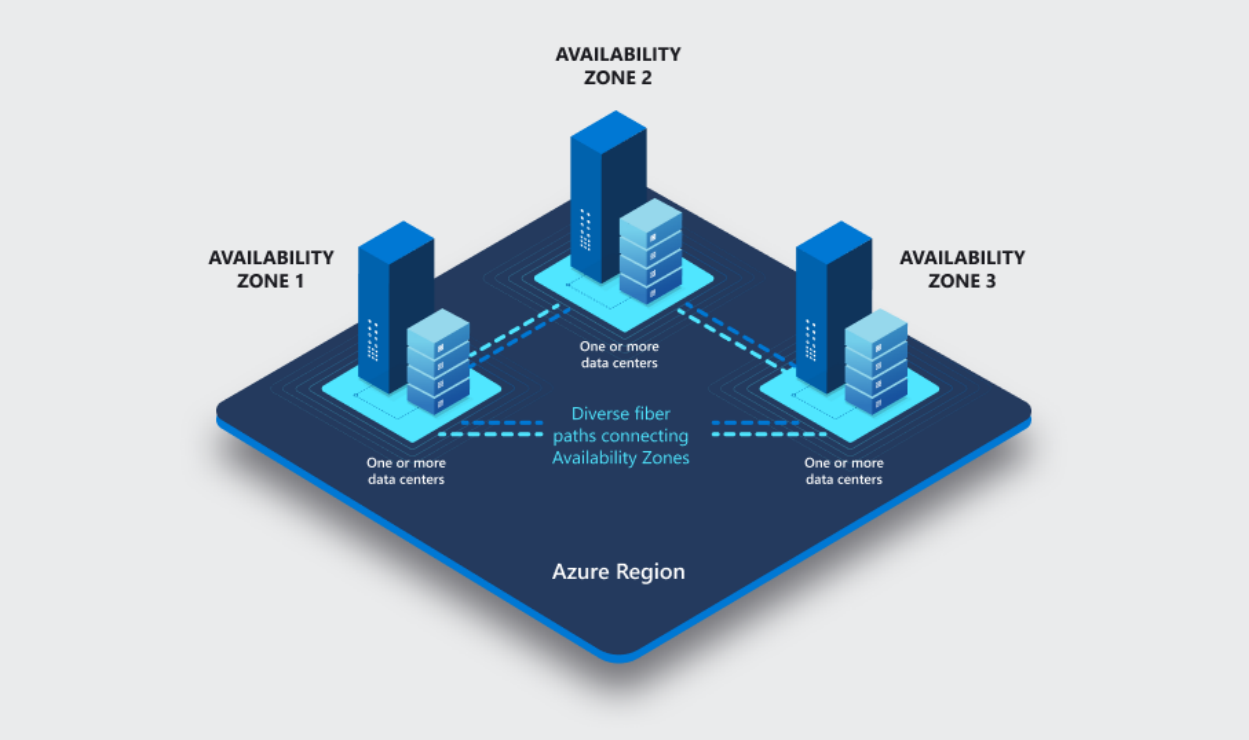
Availability of a Search Solution

The first and easiest way to improve the availability of your search solution is to increase the number of replicas. The only option is to have more than one in the paid-for search service tiers.

The Azure AI Search service has availability guarantees based on the number of replicas you've:

- Two replicas guarantee 99.9% availability for your queries
- Three or more replicas guarantee 99.9% availability for both queries and indexing

The second way to add redundancy to your search solution is to use the Availability Zones. This option requires that you use at least a standard tier.



Search Solution Globally Distributed

The most cost-efficient way to architect an Azure AI Search service is in a single resource group and region. If your business priorities are availability and performance, host multiple versions of your search services in different geographical regions. The benefits of this architecture are:

- Protection against failure in a region. Azure AI Search doesn't support instant failover, you would need to handle it manually.
- If you've globally distributed users or apps, locating a search service nearer to them will improve response times.

Back-Up Options

At present, Azure doesn't offer a formal backup and restore mechanism for Azure AI Search. However, you can build your own tools to back up index definitions as a series of JSON files. Then you can recreate your search indexes using these files.

Monitor a Search Solution

Azure Monitor can give you insights into how well your search service is being used and performing. You can also receive alerts to proactively notify you of issues.

Once you have started using Log Analytics, you get access to performance and diagnostic data in these log tables:

- **AzureActivity** - Shows you tasks that have been executed like scaling the search service
- **AzureDiagnostics** - All the query and indexing operations
- **AzureMetrics** - Data used for metrics that measure the health and performance of your search service

Alerts can let you proactively manage your search service. Here are some commonly used alerts you should consider creating:

- Search Latency
- Throttled search percentage
- Delete Search Service
- Stop Search Service

Semantic Rankings

Semantic ranking is a capability within Azure AI Search that aims to improve the ranking of search results. Semantic ranking improves the ranking of search results by using language understanding to more accurately match the context of the original query.

> 💡 Azure AI Search uses the BM25 ranking function, by default. The BM25 ranking function ranks search results based on the frequency that the search term appears within a document.

Semantic ranking has two functions; it improves the ranking of the query results based on language understanding and it improves the response to the query by providing captions and answers in the results.

Semantic ranking uses the BM25 ranking and calculates a new relevance score using the original BM25 ranking combined with language understanding models to extract the context and meaning of the query.

Semantic Captions and Answers

Semantic captions extract summary sentences from the document verbatim and highlight the most relevant text in the summary sentences.

Semantic answers is an optional additional feature of semantic ranking that provides answers to questions. If the search query appears to be a question and the search results contains text that appears to be a relevant answer, then the semantic answer is returned.

How does it work?

Semantic ranking takes the top 50 results from the BM25 ranking results. The results are split into multiple fields as defined by a semantic configuration. The fields are converted into text strings and trimmed to 256 unique tokens. A token is roughly equivalent to a word in the document.

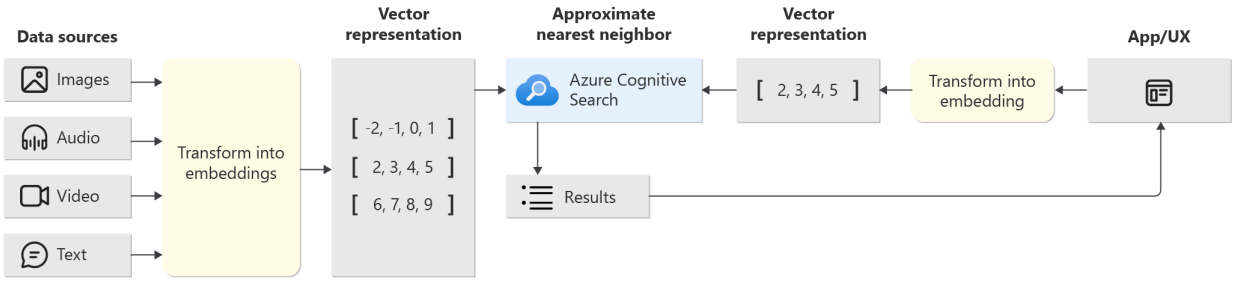
Once the strings are prepared, they are passed to machine reading comprehension models to find the phrases and sentences that best match the query. The results of this summarization phrase is a semantic caption and, optionally, a semantic answer.

The semantic captions are now ranked based on the semantic relevance of the caption. The results are then returned in descending order of relevance.

Vector Search and Retrieval

Vector search is a new capability available in AI Search used to index, store and retrieve vector embedding from a search index. You can use it to power applications implementing the Retrieval Augmented Generation (RAG) architecture, similarity and multi-modal searches or recommendation engines.

Indexing and query worfklows for vector search:



A vector query can be used to match criteria across different types of source data by providing a mathematical representation of the content generated by machine learning models.

There are a few limitations when using vector search, which you should note:

- You'll need to provide the embeddings using Azure OpenAI or a similar open source solution, as Azure AI Search doesn't generate these for your content.
- Customer Managed Keys (CMK) aren't supported.
- There are storage limitations applicable so you should check what your service quota provides.

Embeddings

An embedding is type of data representation that is used by machine learning models. An embedding represents the semantic meaning of a piece of text. You

can visualize an embedding as an array of numbers, and the numerical distance between two embeddings represents their semantic similarity.

There are models specifically created to perform a specific task well.
Use **Similarity** search embeddings to capture the semantic similarity between pieces of text; a **Text** search embedding can look at the relevance of a long document to a short query; use embedding code snippets and natural language search queries using a **Code** search embedding.

Embedding Space

Embedding space is the core of vector queries comprising all the vector fields from the same embedding model. It comprises of all the vector fields populated using the same model.

In this embedding space, similar items are located close together, and dissimilar items are located farther apart.

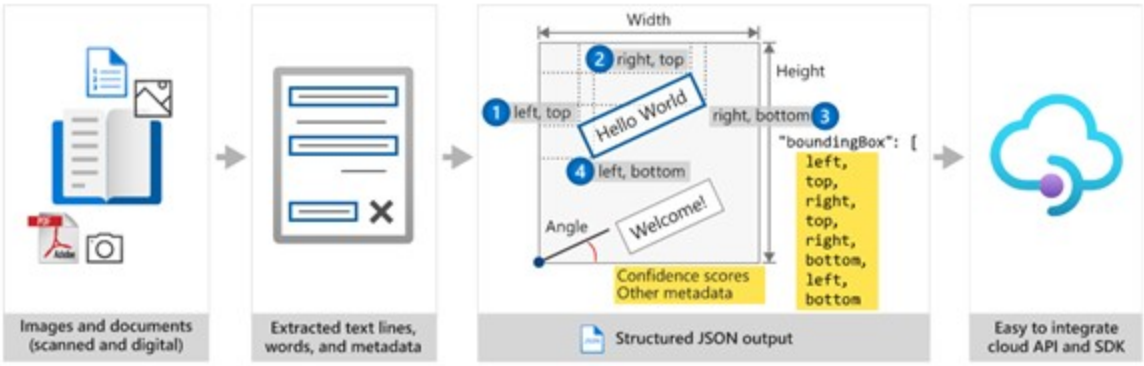
Azure AI Document Intelligence

Azure AI Document Intelligence uses Azure AI Services to analyze the content of scanned forms and convert them into data. It can recognize text values in both common forms and forms that are unique to your business.

Azure AI Document Intelligence outputs data in JSON format, which is widely compatible with many databases, other storage locations, and programming languages.

When you write an application that uses Azure AI Document Intelligence, you need two pieces of information to connect to the resource:

- **Endpoint.** This is the URL where the resource can be contacted.
- **Access key.** This is unique string that Azure uses to authenticate the call to Azure AI Document Intelligence.



Prebuilt Models

In Azure AI Document Intelligence, three of the prebuilt models are for general document analysis:

- **Read:** Use this model to extract words and lines from both printed and hand-written documents. It also detects the language used in the document. For multi-page PDF or TIFF files, you can use the `pages` parameter in your request to fix a page range for the analysis.
- **General document:** The general document model extends the functionality of the read model by adding the detection of key-value pairs, entities, selection marks, and tables
- **Layout:** 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.

> 💡 Selection marks record checkboxes and radio buttons and include whether they're selected or not.

The other prebuilt models expect a common type of form or document:

- Invoice
- Receipt

- W-2 US tax declaration
- ID Document
- Business card
- Health insurance card

If you're using the **Standard** tier, and find your requests are being throttled, you can submit an Azure Support Request to have the default limits increased. The **Free** tier is not available if you are using a multi-service resource.

Features of Prebuilt Models

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.** Spans of text within a document that identify a label or key and its response or value are extracted by many models 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.** Spans of text that indicate a choice can be extracted by some models 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.

Input Requirements

The prebuilt models are very 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 × 50 pixels and 10,000 × 10,000 pixels.
- PDF documents must have dimensions less than 17 × 17 inches or A3 paper size.
- PDF documents must not be protected with a password.
- The total size of the training data set must be 500 pages or less.

> 💡 PDF and TIFF files can have any number of pages but, in the standard tier, only the first 2000 pages are analyzed. In the free tier, only the first two pages are analyzed.

Custom Models

To train a custom model, you must supply at least five examples of the completed form but the more examples you supply, the greater the confidence levels Azure AI Document Intelligence will return when it analyzes input.

Custom Template Models

Custom template model is most appropriate when the forms you want to analyze have a consistent visual template. If you remove all the user-entered data from the forms and find that the blank forms are identical, use a custom template model.

Custom Neural Models

A custom neural model can work across the spectrum of structured to unstructured documents. Documents like contracts with no defined structure or highly structured forms can be analyzed with a neural model. Neural models work on English with the highest accuracy and a marginal drop in accuracy for other languages.

Composed Models

A composed model is one that consists of multiple custom models. Typical scenarios where composed models help are when you don't know the submitted document type and want to classify and then analyze it. They are also useful if you have multiple variations of a form, each with a trained individual model.

The results from a composed model include the `docType` property, which indicates the custom model that was chosen to analyze each form.

Once you've created a set of custom models, you must assemble them into a composed model. You can do this in a Graphical User Interface (GUI) by using Azure AI Document Intelligence Studio, or by using the `StartCreateComposedModelAsync()` method in custom code.

Type of model	Maximum number in Free (F0) tier	Maximum number in Standard (S0) tier
Custom Template	500	5000
Custom Neural	100	500
Composed	5	200

The maximum number of custom models that can be added to a single composed model is 100.

Create a Composed Model in Code

If you're using one of the Azure AI Document Intelligence SDKs to create a composed model by executing code, you have to start by creating an instance of the `DocumentModelAdministrationClient` object, and connecting it to Azure AI Document Intelligence with its endpoint and API key.

To create the composed model, assemble the model IDs of all the custom models in a `List`, and then pass that list to the `StartCreateComposedModelAsync()` method.

In the results, use the `docType` property to determine the constituent custom model that was used to analyze each document.

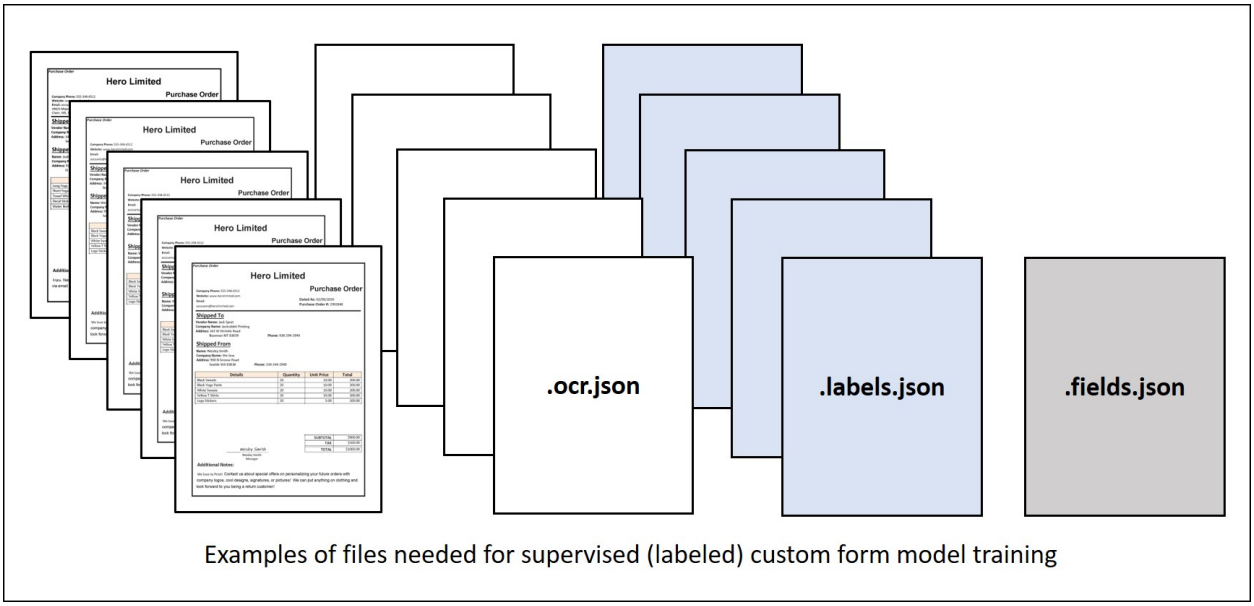
Training Custom Models

To train a custom model:

1. 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.
2. Generate a shared access security (SAS) URL for the container.
3. Use the **Build model** REST API function (or equivalent SDK method).
4. Use the **Get model** REST API function (or equivalent SDK method) to get the trained **model ID**.

OR

1. 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.

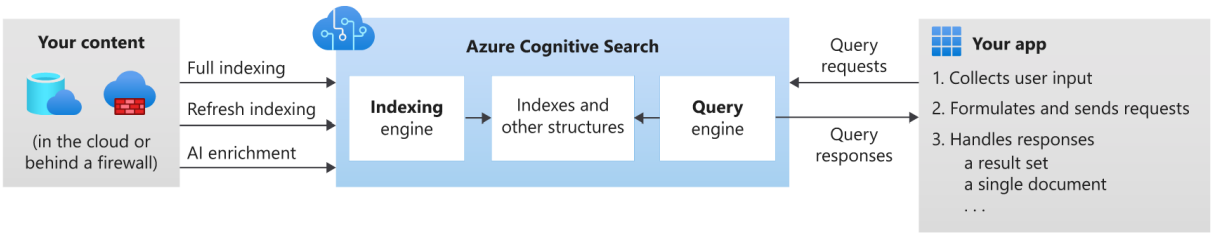


Examples of files needed for supervised (labeled) custom form model training

Intergration with Azure AI Search

If you integrate AI Search with an Azure AI Document Intelligence solution, you can enrich your index with fields that your Azure AI Document Intelligence models are trained to extract.

Azure AI Search is a search service hosted in Azure that can index content on your premises or in a cloud location.



During the indexing process, AI Search crawls your content, processes it, and creates a list of words that will be added to the index, together with their location. There are five stages to the indexing process:

1. **Document Cracking.** In document cracking, the indexer opens the content files and extracts their content.

2. **Field Mappings.** Fields such as titles, names, dates, and more are extracted from the content. You can use field mappings to control how they're stored in the index.
3. **Skillset Execution.** In the optional skillset execution stage, custom AI processing is done on the content to enrich the final index.
4. **Output field mappings.** If you're using a custom skillset, its output is mapped to index fields in this stage.
5. **Push to index.** The results of the indexing process are stored in the index in Azure AI Search.

If you're writing a web service to integrate into a AI Search indexing pipeline, you must conform to certain requirements. For example:

- The service must accept a JSON payload as an input and return a second JSON payload as its results.
- The Output JSON should have a top-level entity named **values** that contains an array of objects.
- The number of objects sent to the service should match the number of objects in the **values** entity.
- Each object in **values** should include a unique **recordId** property, a **data** property with the returned information, a warnings property, and an errors property.

To integrate Azure AI Document Intelligence into the AI Search indexing pipeline, you must:

- Create an Azure AI Document Intelligence resource in your Azure subscription.
- Configure one or more models in Azure AI Document Intelligence.
- Develop and deploy a web service that can call your Azure AI Document Intelligence resource. In this module, you'll use an Azure Function to host this service.

- Add a custom web API skill, with the correct configuration to the AI Search skillset. This skill should be configured to send requests to the web service.

Custom Skill Interface and Requirements

Your code should handle the following input values in the JSON body of the REST request:

- `values`. The JSON body will include a collection named `values`. Each item in this collection represents a form to analyze.
  - `recordId`. Each item in the `values` collection has a `recordId`. You must include this ID in the output JSON so that AI Search can match input forms with their results.
  - `data`. Each item in the `values` includes a `data` collection with two values:
    - `formUrl`. This is the location of the form to analyze.
    - `formSasToken`. If the form is stored in Azure Storage, this token enables your code to authenticate with that account.

From the input data, your code can formulate requests to send to Azure AI Document Intelligence. You'll need the following connection information for these requests:

- The Azure AI Document Intelligence endpoint.
- The Azure AI Document Intelligence API key.

Your code should formulate a REST response that includes a JSON body. The AI Search service expects this response to include:

- `values`. A collection where each item is one of the submitted forms.
  - `recordId`. AI Search uses this value to match results to one of the input forms.
  - `data`. Use the `data` collection to return the fields that Azure AI Document Intelligence has extracted from each input form.
  - `errors`. If you couldn't obtain the analysis for a form, use the `errors` collection to indicate why.

- `warnings`, If you have obtained results but some noncritical problem has arisen, use the `warnings` collection to report the issue.

Azure OpenAI Service

Azure OpenAI Service brings the generative AI models developed by OpenAI to the Azure platform, enabling you to develop powerful AI solutions that benefit from the security, scalability, and integration of services provided by the Azure cloud platform.

To create an Azure OpenAI Service resource from the CLI, refer to this example and replace the following variables with your own:

- MyOpenAIResource: *replace with a unique name for your resource*
- OAIResourceGroup: *replace with your resource group name*
- eastus: *replace with the region to deploy your resource*
- subscriptionID: *replace with your subscription ID*

```
az cognitiveservices account create \
-n MyOpenAIResource \
-g OAIResourceGroup \
-l eastus \
--kind OpenAI \
--sku s0 \
--subscription subscriptionID
```

Azure OpenAI includes several types of model:

- **GPT-4 models** are the latest generation of *generative pretrained* (GPT) models that can generate natural language and code completions based on natural language prompts.
- **GPT 3.5 models** can generate natural language and code completions based on natural language prompts. In particular, **GPT-35-turbo** models are

optimized for chat-based interactions and work well in most generative AI scenarios.

- **Embeddings models** convert text into numeric vectors, and are useful in language analytics scenarios such as comparing text sources for similarities.
- **DALL-E models** are used to generate images based on natural language prompts. Currently, DALL-E models are in preview. DALL-E models aren't listed in the Azure AI Studio interface and don't need to be explicitly deployed.

Deploy AI Models

You can deploy any number of deployments in one or multiple Azure OpenAI resources as long as their Tokens Per Minute (TPM) stays within the deployment quota.

You can also deploy a model using the console. Using this example, replace the following variables with your own resource values:

- OAIResourceGroup: *replace with your resource group name*
- MyOpenAIResource: *replace with your resource name*
- MyModel: *replace with a unique name for your model*
- gpt-35-turbo: *replace with the base model you wish to deploy*

```
az cognitiveservices account deployment create \
   -g OAIResourceGroup \
   -n MyOpenAIResource \
   --deployment-name MyModel \
   --model-name gpt-35-turbo \
   --model-version "0301"  \
   --model-format OpenAI \
   --sku-name "Standard" \
   --sku-capacity 1
```

Test Models in Azure AI Studio’s Playground

Playgrounds are useful interfaces in Azure AI Studio that you can use to experiment with your deployed models without needing to develop your own client application. Azure AI Studio offers multiple playgrounds with different parameter tuning options.

Completion Playground Parameters

There are many parameters that you can adjust to change the performance of your model:

- **Temperature**: Controls randomness. Lowering the temperature means that the model produces more repetitive and deterministic responses. Increasing the temperature results in more unexpected or creative responses. Try adjusting temperature or Top P but not both.
- **Max length (tokens)**: Set a limit on the number of tokens per model response. The API supports a maximum of 4000 tokens shared between the prompt (including system message, examples, message history, and user query) and the model's response. One token is roughly four characters for typical English text.
- **Stop sequences**: Make responses stop at a desired point, such as the end of a sentence or list. Specify up to four sequences where the model will stop generating further tokens in a response. The returned text won't contain the stop sequence.
- **Top probabilities (Top P)**: Similar to temperature, this controls randomness but uses a different method. Lowering Top P narrows the model’s token selection to likelier tokens. Increasing Top P lets the model choose from tokens with both high and low likelihood. Try adjusting temperature or Top P but not both.
- **Frequency penalty**: Reduce the chance of repeating a token proportionally based on how often it has appeared in the text so far. This decreases the likelihood of repeating the exact same text in a response.
- **Presence penalty**: Reduce the chance of repeating any token that has appeared in the text at all so far. This increases the likelihood of introducing new topics in a response.
- **Pre-response text**: Insert text after the user’s input and before the model’s response. This can help prepare the model for a response.

- **Post-response text**: Insert text after the model's generated response to encourage further user input, as when modeling a conversation.

Chat Playground

The Chat playground is based on a conversation-in, message-out interface. You can initialize the session with a system message to set up the chat context.

In the Chat playground, you're able to add *few-shot examples*. The term few-shot refers to providing a few of examples to help the model learn what it needs to do. You can think of it in contrast to zero-shot, which refers to providing no examples.

The Chat playground, like the Completions playground, also includes parameters to customize the model's behavior. The Chat playground also supports other parameters *not* available in the Completions playground. These include:

- **Max response**: Set a limit on the number of tokens per model response. The API supports a maximum of 4000 tokens shared between the prompt (including system message, examples, message history, and user query) and the model's response. One token is roughly four characters for typical English text.
- **Past messages included**: Select the number of past messages to include in each new API request. Including past messages helps give the model context for new user queries. Setting this number to 10 will include five user queries and five system responses.

Integrate Azure OpenAI Service into an App

Azure OpenAI can be accessed via a REST API or an SDK available for Python, C#, JavaScript, and more.

The available endpoints are:

- **Completion** - model takes an input prompt, and generates one or more predicted completions. You'll see this playground in the studio, but won't be covered in depth in this module.
- **ChatCompletion** - model takes input in the form of a chat conversation (where roles are specified with the message they send), and the next chat completion

is generated.

- **Embeddings** - model takes input and returns a vector representation of that input.

> 💡 The `ChatCompletion` endpoint enables the ChatGPT model to have a more realistic conversation by sending the history of the chat with the next user message.

REST API

For each call to the REST API, you need the endpoint and a key from your Azure OpenAI resource, and the name you gave for your deployed model.

Placeholder name	Value
`YOUR_ENDPOINT_NAME`	This base endpoint is found in the **Keys & Endpoint** section in the Azure portal. It's the base endpoint of your resource, such as `https://sample.openai.azure.com/`.
`YOUR_API_KEY`	Keys are found in the **Keys & Endpoint** section in the Azure portal. You can use either key for your resource.
`YOUR_DEPLOYMENT_NAME`	This deployment name is the name provided when you deployed your model in the Azure AI Studio.

REST endpoints allow for specifying other optional input parameters, such as `temperature`, `max_tokens` and more. If you'd like to include any of those parameters in your request, add them to the input data with the request.

Embeddings

Embeddings are helpful for specific formats that are easily consumed by machine learning models. To generate embeddings from the input text, `POST` a request to the `embeddings` endpoint.

When generating embeddings, be sure to use a model in Azure OpenAI meant for embeddings. Those models start with `text-embedding` or `text-similarity`, depending on what functionality you're looking for.

Prompt Engineering

Prompt engineering is the process of designing and optimizing prompts to better utilize AI models. Designing effective prompts is critical to the success of prompt engineering, and it can significantly improve the AI model's performance on specific tasks. Providing relevant, specific, unambiguous, and well structured prompts can help the model better understand the context and generate more accurate responses.

In particular, `temperature` and `top_p` (top_probability) are the most likely to impact a model's response as they both control randomness in the model, but in different ways. It's recommended to change either `temperature` or `top_p` at a time, but not both.

Primary, Supporting, and Grounding Content

A specific technique for formatting instructions is to split the instructions at the beginning or end of the prompt, and have the user content contained within `---` or `###` blocks. These tags allow the model to more clearly differentiate between instructions and content. For example:

```
Translate the text into French

---
What's the weather going to be like today?
---
```

- Primary content refers to content that is the subject of the query, such as a sentence to translate or an article to summarize. This content is often included at the beginning or end of the prompt (as an instruction and differentiated by `---` blocks), with instructions explaining what to do with it.
- Supporting content is content that may alter the response, but isn't the focus or subject of the prompt. Examples of supporting content include things like names, preferences, future date to include in the response, and so on.
- Grounding content allows the model to provide reliable answers by providing content for the model to draw answer from. Grounding content could be an

essay or article that you then ask questions about, a company FAQ document, or information that is more recent than the data the model was trained on.

Cues

Cues are leading words for the model to build upon, and often help shape the response in the right direction. Cues are particularly helpful if prompting the model for code generation.

Additional Ways to Improve Accuracy

a. Request output composition: Specifying the structure of your output can have a large impact on your results. This could include something like asking the model to cite their sources, write the response as an email, format the response as a SQL query, classify sentiment into a specific structure, and so on.

b. System message: The system message is included at the beginning of a prompt and is designed to give the model instructions, perspective to answer from, or other information helpful to guide the model's response. This system message might include tone or personality, topics that shouldn't be included, or specifics (like formatting) of how to answer.

   The `ChatCompletion` endpoint enables including the system message by using the `System` chat role.

c. Conversation history: Conversation history enables the model to continue responding in a similar way (such as tone or formatting) and allow the user to reference previous content in subsequent queries. This history can be provided in two ways: from an actual chat history, or from a user defined example conversation.

   In the **Parameters** section of the chat playground, you can specify how many past messages you want included. Try reducing that to 1 or increasing to max to see how different amounts of history impact the conversation.

More conversation history included in the prompt means a larger number of input tokens are used. You will have to determine what the correct balance is for your use case, considering the token limit of the model you are using.

d. Few shot learning: Using a user defined example conversation is what is called *few shot learning*, which provides the model examples of how it should respond to a given query. These examples serve to train the model how to respond.

e. Chain of thought: Asking a model to respond with the step by step process by which it determined the response is a helpful way to understand how the model is interpreting the prompt.

DALL-E

DALL-E is a neural network-based model that can generate graphical data from natural language input.

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

To experiment with DALL-E, you can provision an Azure OpenAI Service resource in an Azure subscription and use the *Images playground* to submit prompts and view the resulting generated images.

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

- The resolution (size) of the generated images. Available sizes are `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`).

Consume DALL-E Models with REST API

The request must contain the following parameters in a JSON body:

- **prompt**: The description of the image to be generated.
- **n**: The number of images to be generated. DALL-E 3 only supports n=1.
- **size**: The resolution of the image(s) to be generated (*1024×1024*, *1792×1024*, or *1024×1792*).
- **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*.

```
{
    "prompt": "A badger wearing a tuxedo",
    "n": 1,
    "size": "512x512",
    "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 used>"
        }
    ]
}
```

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.

Implement Retrieval Augmented Generation (RAG)

RAG with Azure OpenAI allows developers to use supported AI chat models that can reference specific sources of information to ground the response. Adding this information allows the model to reference both the specific data provided and its pretrained knowledge to provide more effective responses.

Azure OpenAI enables RAG by connecting pretrained models to your own data sources. Azure OpenAI on your data utilizes the search ability of Azure AI Search to add the relevant data chunks to the prompt. Once your data is in a AI Search index, Azure OpenAI on your data goes through the following steps:

1. Receive user prompt.
2. Determine relevant content and intent of the prompt.
3. Query the search index with that content and intent.
4. Insert search result chunk into the Azure OpenAI prompt, along with system message and user prompt.
5. Send entire prompt to Azure OpenAI.
6. Return response and data reference (if any) to the user.

Fine-Tuning vs RAG

Fine-tuning is a technique used to create a custom model by training an existing foundational model such as `gpt-35-turbo` with a dataset of additional training data. Fine-tuning can result in higher quality requests than prompt engineering alone, customize the model on examples larger than can fit in a prompt, and allow the user to provide fewer examples to get the same high quality response. However, the process for fine-tuning is both costly and time intensive, and should only be used for use cases where it's necessary.

RAG with Azure OpenAI on your data still uses the stateless API to connect to the model, which removes the requirement of training a custom model with your data and simplifies the interaction with the AI model. AI Search first finds the useful information to answer the prompt, adds that to the prompt as grounding data, and Azure OpenAI forms the response based on that information.

Add a Data Source

Adding your data is done through the Azure AI Studio, in the **Chat** playground, or by specifying your data source in an API call. The data source you add is then used to augment the prompt sent to the model. When setting up your data in the studio, you can choose to upload your data files, use data in a blob storage account, or connect to an existing AI Search index.

If you're uploading or using files already in a storage account, Azure OpenAI on your data supports `.md`, `.txt`, `.html`, `.pdf`, and Microsoft Word or PowerPoint files.

Using API

The system message, for example, is a useful reference for instructions for the model and is included with every call. While there's no token limit for the system message, when using your own data the system message gets truncated if it exceeds the model's token limit (which varies per model, from 400 to 4000 tokens). The response from the model is also limited when using your own data is 1500 tokens.

Due to these token limitations, it's recommended that you limit both the question length and the conversation history length in your call.

Using the API with your own data, you need to specify the data source where your data is stored. With each call you need to include the `endpoint`, `key`, and `indexName` for your AI Search resource.

The call when using your own data needs to be sent to a different endpoint than is used when calling a base model, which includes `extensions`. Your call will be sent to a URL similar to the following.

```
<your_azure_openai_resource>/openai/deployments/<deployment_name>/chat/completions?api-version=<version>
```

Responsible Generative AI

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. *Identify* 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. *Operate* the solution responsibly by defining and following a deployment and operational readiness plan.