

AI-102 Notes

Chapter 1:

responsible ai:

1. fairness
2. reliability and safety
3. privacy and security
4. Inclusiveness
5. Transparency
6. Accountability

Natural language processing	Knowledge mining and document intelligence	Computer vision	Decision support	Generative AI
Text analysis	AI Search	Image analysis	Content safety	Azure OpenAI Service
Question answering	Document Intelligence	Video analysis	Content moderation	DALL-E image generation
Language understanding	Custom Document Intelligence	Image classification		
Translation	Custom skills	Object detection		
Named entity recognition		Facial analysis		
Custom text classification		Optical character recognition		
Speech		Azure AI Video Indexer		
Speech Translation				

Use Azure AI Search to index documents for search.

Language	Speech	Vision	Decision
Azure AI Language	Azure AI Speech	Azure AI Computer Vision	Azure AI Anomaly Detector
Azure AI Translator		Azure AI Custom Vision	Azure AI Content Moderator
		Azure AI Face	Azure AI Personalizer

You can use AI services to build your own AI solutions to provide out-of-the-box solutions for common AI scenarios. Azure AI services include:

- **Azure AI Document Intelligence** - An optical character recognition (OCR) solution that can extract semantic meaning from forms, such as invoices, receipts, and others.

- **Azure AI Immersive Reader** - A reading solution that supports people of all ages and abilities.
- **Azure Cognitive Search** - A cloud-scale search solution that uses AI services to extract insights from data and documents.
- **Azure OpenAI** - An Azure Cognitive Service that provides access to the capabilities of OpenAI GPT-4.

- multi-service resource - same endpoint for all services
- single-service resource - usually free , diff end point for diff services

To consume the service through the endpoint

- endpoint URI
- subscription key
- the resource location - some require location also

to interact and integrate azure ai service we use REST APIs and SDK

REST APIs - data is submitted in JSON format over an HTTP req . with POST PUT GET, the result is returned to the client as an HTTP response often with JSON contents

- GET - to retrieve data
- PUT - to update an existing data
- POST - to send data to the server

You can regenerate keys using the Azure portal, or using the `az cognitiveservices account keys regenerate` Azure command-line interface (CLI) command.

access to keys is given to security principals, administrators assign a security principal to an application then it is called as service principal , and for that specific application it is called as managed identity(access role for that application) for the application.

you assign a role to service principal

CLI command to list number of keys:

```
az cognitiveservices account keys list --name <resourceName> --resource-group <resourceGroup>
```

using key1 and CLI to analyse text

```
curl -X POST "<yourEndpoint>/language/:analyze-text?api-version=2023-04-01" -H "Content
```

```
regenerate keys
```

```
az cognitiveservices account keys regenerate --name <resourceName> --resource-group <re
```

alert rule is created on the resource you want to monitor

monitor costs - costs analysis tab

alert for a resource - alert tab

metrics - metrics page

azure log analytics - query and visualize log data

azure storage - used to store log archives

A container comprises an application or service and the runtime components needed to run it,

- Containers are portable across hosts, which may be running different operating systems or use different hardware - making it easier to move an application and all its dependencies.
- A single container host can support multiple isolated containers, each with its own specific runtime configuration - making it easier to consolidate multiple applications that have different configuration requirement.

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.

container images : <https://learn.microsoft.com/en-gb/training/modules/investigate-container-for-use-with-ai-services/3-use-ai-services-container>

container config :

ApiKey

Billing

Eula

- **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.

Chapter 2:

AI Vision Service :

- *Description and tag generation* - determining an appropriate caption for an image, and identifying relevant "tags" that can be used as keywords to indicate its subject.
- *Object detection* - detecting the presence and location of specific objects within the image.
- *People detection* - detecting the presence, location, and features of people in the image.
- *Image metadata, color, and type analysis* - determining the format and size of an image, its dominant color palette, and whether it contains clip art.
- *Category identification* - identifying an appropriate categorization for the image, and if it contains any known landmarks.
- *Background removal* - detecting the background in an image and output the image with the background transparent or a greyscale alpha matte image.
- *Moderation rating* - determine if the image includes any adult or violent content.
- *Optical character recognition* - reading text in the image.
- *Smart thumbnail generation* - identifying the main region of interest in the image to create a smaller "thumbnail" version.

Analyze an Image :

```
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.READ],
    gender_neutral_caption=True,
    language="en",
)
```

- `VisualFeatures.Tags`: Identifies tags about the image, including objects, scenery, setting, and actions
- `VisualFeatures.Objects`: Returns the bounding box for each detected object
- `VisualFeatures.Caption`: Generates a caption of the image in natural language

- `VisualFeatures.DenseCaptions`: Generates more detailed captions for the objects detected
- `VisualFeatures.People`: Returns the bounding box for detected people
- `VisualFeatures.SmartCrops`: Returns the bounding box of the specified aspect ratio for the area of interest
- `VisualFeatures.Read`: Extracts readable text

COCO file is present in a storage blob storage container and is created when you label the images present in the container.

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.

two ai services for face detection

The Azure AI Vision service

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

The Face service

The **Face** service offers more comprehensive facial analysis capabilities than the Azure AI Vision service, including:

- Face detection (with bounding box).
- Comprehensive facial feature analysis (including head pose, presence of spectacles, blur, facial landmarks, occlusion, and others).
- Face comparison and verification.
- Facial recognition.

while building facial solution responsible ai's:

- data privacy and security
- transparency

- fairness and inclusiveness

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.

Subsequently ID is matched

- *Face attribute analysis* - you can return a wide range of facial attributes, including:
 - Head pose (*pitch*, *roll*, and *yaw* orientation in 3D space)
 - Glasses (*NoGlasses*, *ReadingGlasses*, *Sunglasses*, or *Swimming Goggles*)
 - Blur (*low*, *medium*, or *high*)
 - Exposure (*underExposure*, *goodExposure*, or *overExposure*)
 - Noise (visual noise in the image)
 - Occlusion (objects obscuring the face)
 - Accessories (glasses, headwear, mask)
 - QualityForRecognition (*low*, *medium*, or *high*)

OCR uses AI Vision Service

two different features that read text from documents:

- Image Analysis -OCR(mainly for unstructured data , handwritten notes)
- Document Intelligence (receipts,articles,invoices)

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

```
https://<endpoint>/computervision/imageanalysis:analyze?features=read&...
```

The **Azure Video Indexer** service is designed to help you extract information from videos. It provides functionality that you can use for:

- *Facial recognition* - detecting the presence of individual people in the image. This requires Limited Access approval.
- *Optical character recognition* - reading text in the video.
- *Speech transcription* - creating a text transcript of spoken dialog in the video.
- *Topics* - identification of key topics discussed in the video.
- *Sentiment* - analysis of how positive or negative segments within the video are.

- *Labels* - label tags that identify key objects or themes throughout the video.
- *Content moderation* - detection of adult or violent themes in the video.
- *Scene segmentation* - a breakdown of the video into its constituent scenes.

Azure Video Indexer provides a REST API that you can use to obtain information about your account, including an access token.

HTTPCopy

```
https://api.videoindexer.ai/Auth/<location>/Accounts/<accountId>/AccessToken
```

You can then use your token to consume the REST API and automate video indexing tasks, creating projects, retrieving insights, and creating or deleting custom models.

For example, a GET call

to `https://api.videoindexer.ai/<location>/Accounts/<accountId>/Customization/CustomLogos/Logos/<logoId>?<accessToken>` REST endpoint returns the specified logo. In another example, you can send a GET request to `https://api.videoindexer.ai/<location>/Accounts/<accountId>/Videos?<accessToken>`, which returns details of videos in your account, similar to the following JSON example:


Chapter 3 (NLP - Azure AI Language Resource)

Provision an Azure AI Language resource

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

- *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.

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).

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

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

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

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

```
detectedLanguage = ai_client.detect_language(documents=[text])[0]
sentimentAnalysis = ai_client.analyze_sentiment(documents=[text])[0]
```

QnA

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.

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

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.

for multi-turn conversation use follow up prompts

to improve qna performance you can use active learning(suggestions) and synonyms(alterations).

natural language understanding

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. Learned features require building and training a model to correctly predict appropriate labels, which is covered in upcoming units of this module.

Pre-configured features

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

The following features are all pre-configured.

Summarization

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

Named entity recognition

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

Personally identifiable information (PII) detection

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

Key phrase extraction

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

Sentiment analysis

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

Language detection

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

Learned features

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

Conversational language understanding (CLU)

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

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

Custom named entity recognition

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

Custom text classification

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

Question answering

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

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

APIs for CLU

<https://learn.microsoft.com/en-gb/training/modules/build-language-understanding-model/2-understand-resources-for-building>

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.

Types of entities

You can split entities into a few different component types:

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

You can have up to five prebuilt components per entity

process for creating a clu

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

all the prebuilt entities

<https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-reference-prebuilt-entities>

Part of NLP is the ability to classify text, and Azure provides ways to classify text including sentiment, language, and custom categories defined by the user.

- 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:

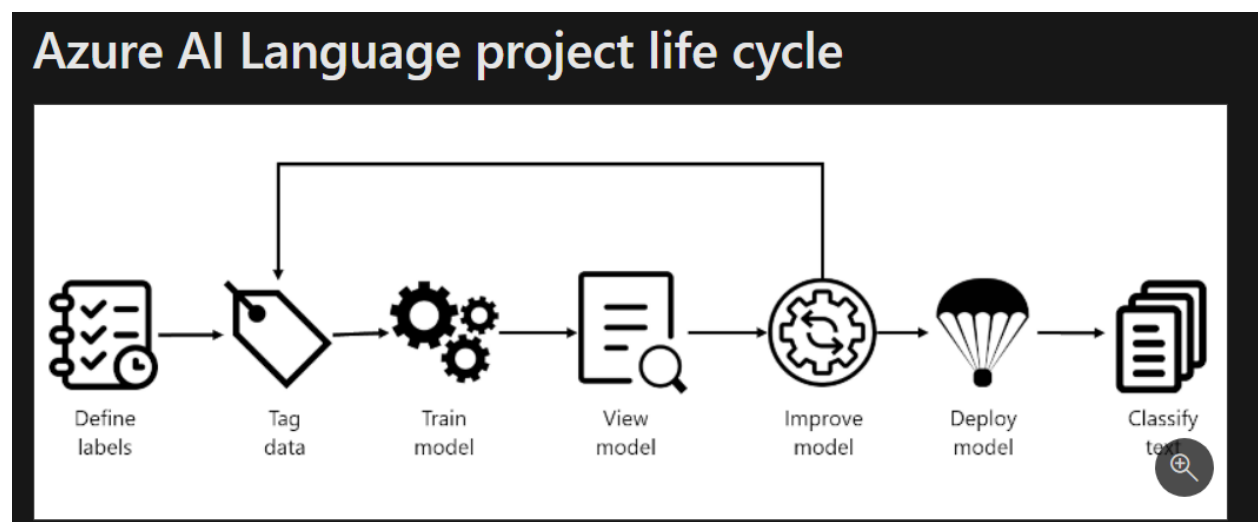
- **Recall** - Of all the actual labels, how many were identified; the ratio of true positives to all that was labeled.
- **Precision** - How many of the predicted labels are correct; the ratio of true positives to all identified positives.
- **F1 Score** - A function of *recall* and *precision*, intended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

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

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

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

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



manual split for smaller dataset
automatic split for larger dataset

custom text classification (apis)

<https://learn.microsoft.com/en-gb/training/modules/custom-text-classification/3-understand-how-to-build-projects>

NER(Named Entity Recognition):

consideration for data selection:

- diversity
- distribution
- accuracy

label your data:

- consistency
- precision
- completeness

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 Azure AI Translator provides an API for translating text between 90 supported languages.

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

- Language detection.
- One-to-many translation.
- Script transliteration (converting text from its native script to an alternative script).

three options in translation:

- Word Alignment (which part of the word corresponds to which part of the source)
- sentence length (len of src and len of translated)
- profanity filter - ()
- profanity filter - (filters out inappropriate language in the translated text - **NoAction**: Profanities are translated along with the rest of the text.
- **Deleted**: Profanities are omitted in the translation.
- **Marked**)

200 successful response code , 400 (invalid or missing query parameters)

azure speech API :

- **Speech to text**: An API that enables *speech recognition* in which your application can accept spoken input.

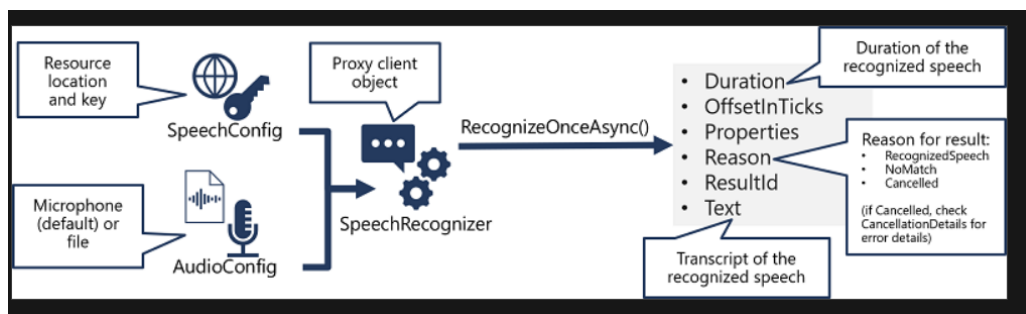
- **Text to speech:** An API that enables *speech synthesis* in which your application can provide spoken output.
- **Speech Translation:** An API that you can use to translate spoken input into multiple languages.
- **Speaker Recognition:** An API that enables your application to recognize individual speakers based on their voice.
- **Intent Recognition:** An API that uses conversational language understanding to determine the semantic meaning of spoken input.

for translation and speech SDK you need location and key

Speech to text APIs

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

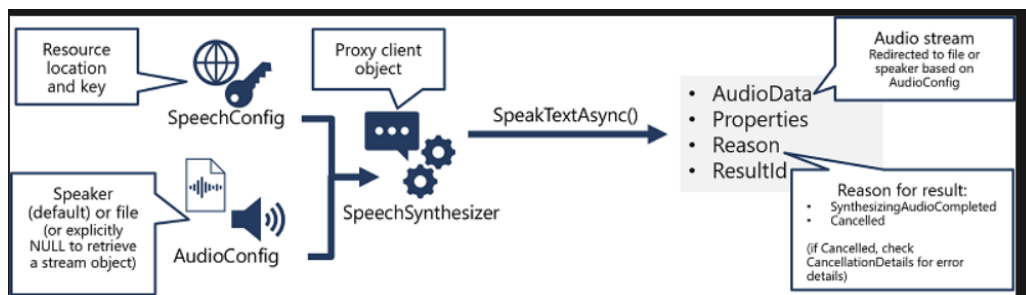
- The **Speech to text** API, which is the primary way to perform speech recognition.
- The **Speech to text Short Audio** API, which is optimized for short streams of audio (up to 60 seconds).



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.



we use the `speechconfig` object to customize the audio that is returned by the speech service

customizations that we can do is :

audio format : (audio file type , sample rate , bit depth)

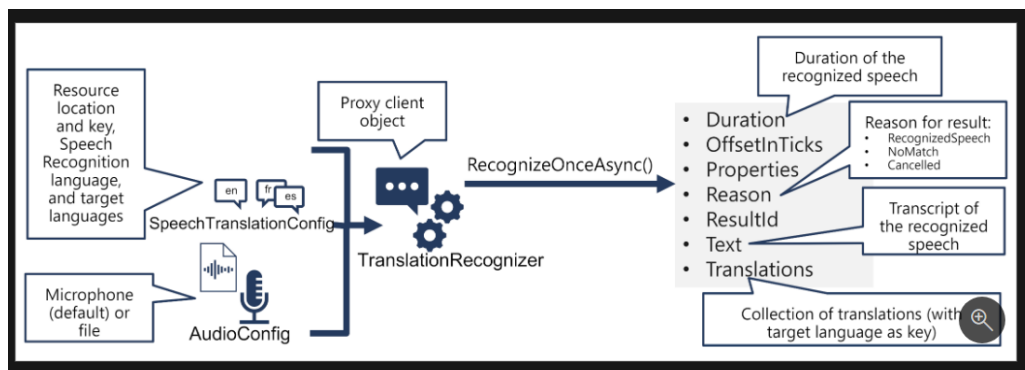
voices (standard & neural)

SSML(speech synthesis markup language)

The service also supports an XML-based syntax for describing the characteristics of the speech you want to generate. This **Speech Synthesis Markup Language** (SSML) syntax offers greater control over how the spoken output sounds, enabling you to:

- Specify a speaking style, such as "excited" or "cheerful" when using a neural voice.
- Insert pauses or silence.
- Specify *phonemes* (phonetic pronunciations), for example to pronounce the text "SQL" as "sequel".
- Adjust the *prosody* of the voice (affecting the pitch, timbre, and speaking rate).
- Use common "say-as" rules, for example to specify that a given string should be expressed as a date, time, telephone number, or other form.
- Insert recorded speech or audio, for example to include a standard recorded message or simulate background noise.

translate speech to text



to synthesize translations :

event based synthesis : 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 :

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

AI Search

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.

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 ($R \times P = SU$). For example, a resource with four replicas and three partitions is using 12 search units.

components in AI search

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:

(each bullet point can be considered as a skill and all combined can be called as a skillset)

- 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.

In some cases, such as when you add new fields to an index or new skills to a skillset, you may need to reset the index before re-running the indexer.

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.

attributes for index field

- **key**: Fields that define a unique key for index records.
- **searchable**: Fields that can be queried using full-text search.
- **filterable**: Fields that can be included in filter expressions to return only documents that match specified constraints.
- **sortable**: Fields that can be used to order the results.
- **facettable**: Fields that can be used to determine values for *facets* (user interface elements used to filter the results based on a list of known field values).
- **retrievable**: Fields that can be included in search results (*by default, all fields are retrievable unless this attribute is explicitly removed*).

Understanding the indexing process:

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).

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.

- document
 - metadata_storage_name
 - metadata_author
 - content
 - normalized_images
 - image0
 - image1
 - language

each skill adds a field to the document like language , and if the skills are applied on the specific context within the hierarchy then it is added below the hierarchy for example OCR attached to the images

- document
 - metadata_storage_name
 - metadata_author
 - content
 - normalized_images
 - image0
 - Text
 - image1
 - Text
 - language

you can merge fields also

search 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.

1. simple : basic search
2. full : filtering , regex

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

- **search** - A search expression that includes the terms to be found.
- **queryType** - The Lucene syntax to be evaluated (*simple* or *full*).
- **searchFields** - The index fields to be searched.
- **select** - The fields to be included in the results.
- **searchMode** - Criteria for including results based on multiple search terms. For example, suppose you search for *comfortable hotel*. A searchMode value of *Any* returns documents that contain "comfortable", "hotel", or both; while a searchMode value of *All* restricts results to documents that contain both "comfortable" and "hotel".

Query processing consists of four stages:

1. *Query parsing*. The search expression is evaluated and reconstructed as a tree of appropriate subqueries. Subqueries might include *term queries* (finding specific individual words in the search expression - for example *hotel*), *phrase queries* (finding multi-term phrases specified in quotation marks in the search

expression - for example, "*free parking*"), and *prefix queries* (finding terms with a specified prefix - for example *air**, which would match *airway*, *air-conditioning*, and *airport*).

2. *Lexical analysis* - The query terms are analyzed and refined based on linguistic rules. For example, text is converted to lower case and nonessential *stopwords* (such as "the", "a", "is", and so on) are removed. Then words are converted to their *root* form (for example, "comfortable" might be simplified to "comfort") and composite words are split into their constituent terms.
3. *Document retrieval* - The query terms are matched against the indexed terms, and the set of matching documents is identified.
4. *Scoring* - A relevance score is assigned to each result based on a term frequency/inverse document frequency (TF/IDF) calculation.

You can apply filters to queries in two ways:

- By including filter criteria in a *simple search* expression.
- By providing an OData filter expression as a **\$filter** parameter with a *full* syntax search expression.

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

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

using facets

```
search=*  
facet=author
```

```
search=*  
$filter=author eq 'selected-facet-value-here'
```

sorting

```
search=*  
$orderby=last_modified desc
```

enhancing the search

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.

After you've added a suggester, you can use the **suggestion** and **autocomplete** REST API endpoints or the .NET **DocumentsOperationsExtensions.Suggest** and **DocumentsOperationsExtensions.Autocomplete** methods to submit a partial search term and retrieve a list of suggested results or autocompleted terms to display in the user interface.

custom scoring

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

using synonym map

use can add custom skills as web hosted services such as Azure Function for example form recognizer

input schema for a custom skill defines a JSON structure

to add skill use this API : **Custom.WebApiSkill**

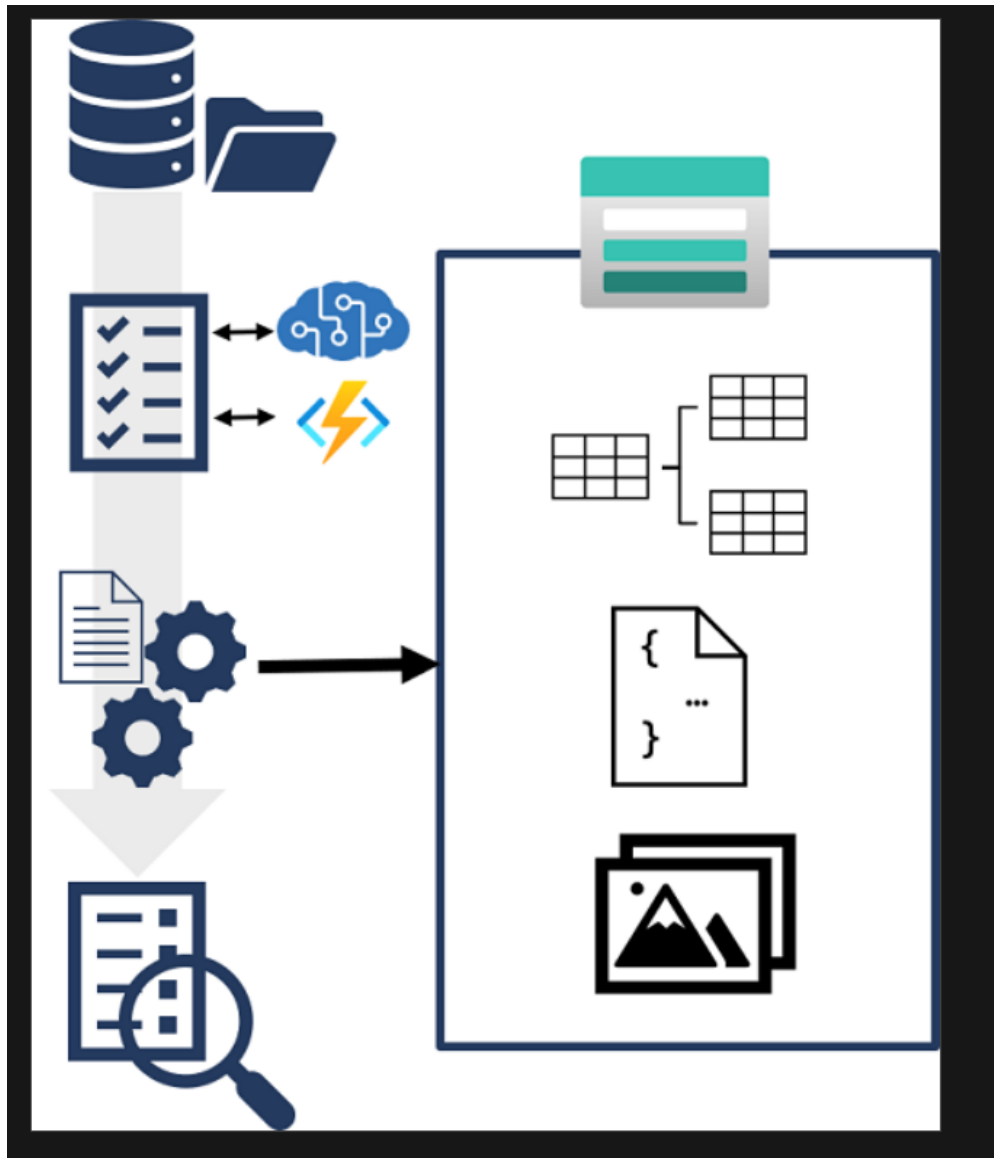
The skill definition must:

- 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 stores :

While the index might be considered the primary output from an indexing process, the enriched data it contains might also be useful in other ways. For example:

- Since the index is essentially a collection of JSON objects, each representing an indexed record, it might be useful to export the objects as JSON files for integration into a data orchestration process using tools such as Azure Data Factory.
- You may want to normalize the index records into a relational schema of tables for analysis and reporting with tools such as Microsoft Power BI.
- Having extracted embedded images from documents during the indexing process, you might want to save those images as files.



Azure AI Search supports these scenarios by enabling you to define a *knowledge store* in the skillset that encapsulates your enrichment pipeline. The knowledge store consists of *projections* of the enriched data, which can be JSON objects, tables, or image files.

We use shaper skill to convert the complex JSON document which is difficult to work with to a new field containing a simpler structure for the fields you want to map to projections .

to create a 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.

however note that you must define a separate *projection* for each type of projection, even though each projection contains lists for tables, objects, and files. Projection types are mutually exclusive in a projection definition, so only one of the projection type lists can be

populated.

For

object and **file** projections, the specified container will be created if it does not already exist. An Azure Storage table will be created for each **table** projection, with the mapped fields and a unique key field with the name specified in the **generatedKeyName** property.

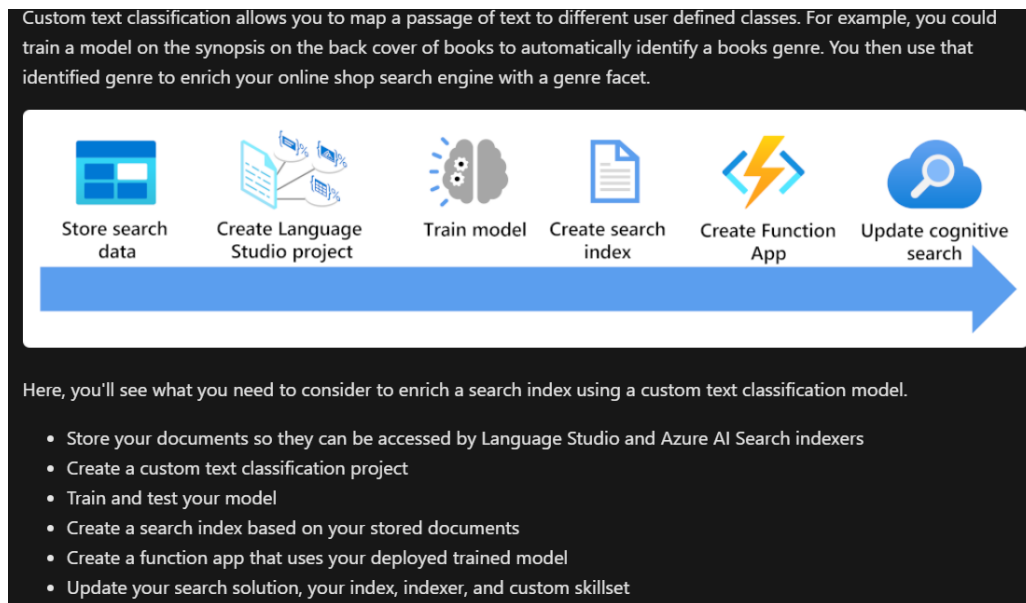
File projections create a .jpg file for each image extracted from a document.

Azure AI Language features

Azure AI Language groups its features into the following areas:

1. Classify text
2. Understand questions and conversational language
3. Extract information
4. Summarize text
5. Translate text

Enrich AI search index with azure AI language



2. Which step enables an AI Search index to store the enrichments from an Azure AI Language project? *

☐ Update your Azure AI Search solution.

✓ Correct. You need to edit the index, indexer, and custom skillset to store the enrichments.

☒ Create a function app.

✗ Incorrect. You do need to create a function app to call the trained model, but you don't store the enrichments in the search index.

☐ Train a custom text classification model.

3. Where do you copy the full endpoint of the function app to use in the custom web skill that includes the api-version and path to the actual function name? *

☒ You can find the endpoint in the Azure portal.

✗ Incorrect. You can find the endpoint for the overall function app in the Azure portal, but this won't give you the full URL to call the specific function.

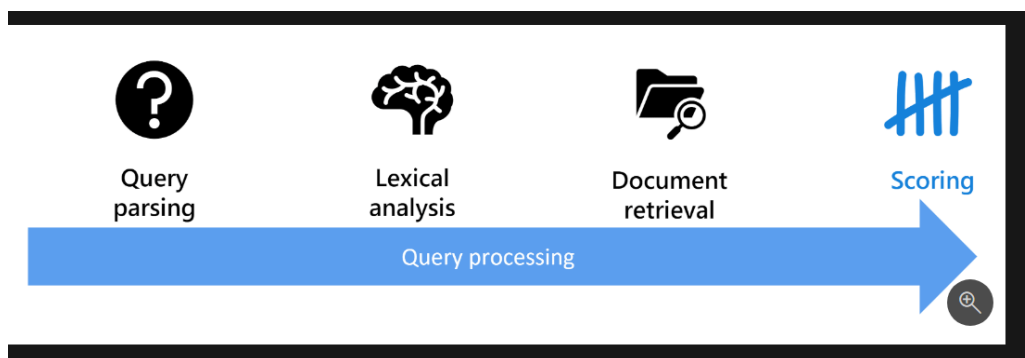
☐ You can find the endpoint in Language Studio.

☐ You can find the endpoint in the Azure extension inside VS Code.

✓ Correct. You right click on the function name in the Azure extension and select Copy function URL.

Lucene Syntax :

<https://learn.microsoft.com/en-gb/training/modules/implement-advanced-search-features-azure-cognitive-search/02-improve-ranking-of-document-term-boosting>



Azure AI Search uses the BM25 similarity ranking algorithm.

As the default scoring works on the frequency of terms and rarity, the final calculated score might not return the highest score for the most relevant document. Each dataset is different, so AI Search lets you influence a document score using scoring profiles.

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

You can control which scoring profile is applied to a search query by appending the `&scoringProfile=PROFILE_NAME` parameter.

functions in scoring profile:

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

Azure AI Search is configured by default to analyze text and identify tokens that will be helpful in your index. The right tokens ensure that users can find the documents they need quickly. In most cases, the default configuration produces an optimal index. However, when you have unusual or unique fields, you might want to configure exactly how text is analyzed.

When AI Search indexes your content, it retrieves text.

To build a useful index, with terms that help users locate documents, that text needs processing. For example:

- The text should be broken into words, often by using whitespace and punctuation characters as delimiters.
- Stopwords, such as "the" and "it", should be removed because users don't search for them.
- Words should be reduced to their root form. For example, past tense words, such as "ran", should be replaced with present tense words, such as "run".

If you don't specify an analyzer for a field, the default Lucene analyzer is used.

- **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 consists of:

- **Character filters.** These filters process a string before it reaches the tokenizer.
- **Tokenizers.** These components divide the text into tokens to be added to the index.
- **Token filters.** These filters remove or modify the tokens emitted by the tokenizer.

types of tokenizers

- **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.

ou can include only one tokenizer but one or more character filters and one or more token filters. Use a unique name for your analyzer and set the `@odata.type` property to `Microsoft.Azure.Search.CustomAnalyzer`.

It's also possible to use a different analyzer when indexing the field and when searching the field.

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 a straight line across the Earth's surface from the point you specify to the location of the search result.
- `geo.intersects`. This function returns `true` if the location of a search result is inside a polygon that you specify.

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.

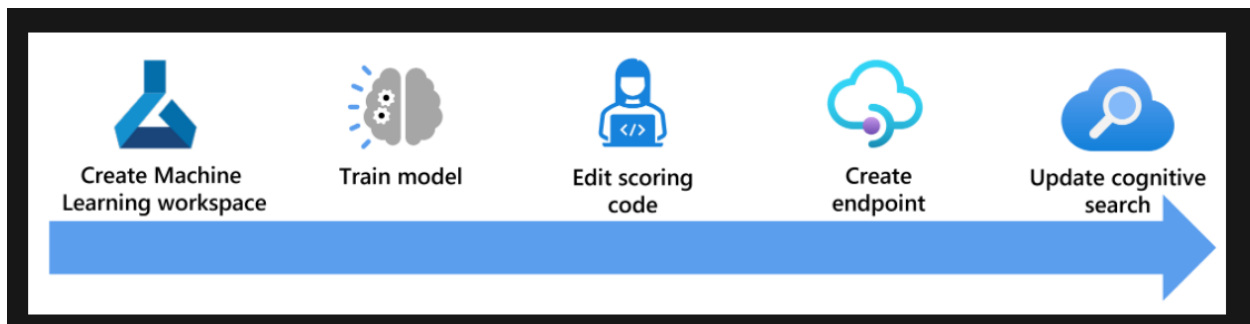
```
search=(Description:luxury OR Category:luxury)&orderby=geo.distance(Location, geography'POINT(2.294481 48.858370)')
asc&$select=HotelId, HotelName, Category, Tags, Description&$count=true
```

```
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
```

`geo.intersects` returns a boolean value, so it's not possible to use it in an `orderby` clause.

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`.

Take note that the custom skill doesn't include settings for `batchSize`. As the AML model will process a single document at a time.



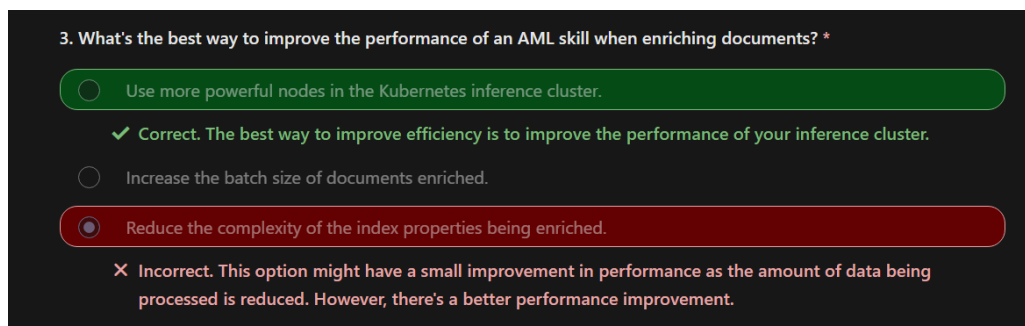
The other restriction is that the endpoint has to be an Azure Kubernetes Service (AKS), container instances aren't supported.

With everything above in place, you need to update your Azure AI Search service. First, to enrich your search index you'll add a new field to your index to include the output for the model.

Then you'll update your index skillset and add the `#Microsoft.Skills.Custom.AmlSkill` custom skill.

Next, you'll change your indexer to map the output from the custom skill to the field you created on the index.

The last step is to rerun your indexer to enrich your index with the AML model.



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.

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.

limitations of using ADF

Azure AI Search data type

String

Int32

Int64

Double

Boolean

DateTimeOffset

This means ComplexTypes and arrays aren't currently supported. Looking at the JSON document above this means that it isn't possible to map all the phone numbers for the customer. Only the first telephone number has been mapped.

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

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

backoff 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.

This code uses asynchronous calls to a function `ExponentialBackoffAsync` that implements the backoff strategy. You call the function using threads, for example, the number of cores your processor has. When the maximum number of threads has been used, the code waits for any thread to finish. It then creates a new thread until all the documents are uploaded.

2. Which feature of the REST API would you use to upload documents into a search index? *

☐ Index.

✓ Correct. You use the index REST API focused on documents.

☒ Indexer.

✗ Incorrect. The indexer is a feature of the search service that pulls data into an index.

☐ Skillset.

AI Search security builds on Azure's existing network security features. 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 in transit is encrypted using the standard HTTPS TLS 1.3 encryption over port 443.

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

You can configure Azure AI Search to restrict the documents someone can search, for example, restrict searching contractual PDFs to people in your legal department.

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.

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 response. If your indexes are being throttled, they'll show up as 207 HTTP responses.

If you know how the search service works, you can tune your queries to drastically improve performance. 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`.

. The largest index supported currently is 12 partitions in the L2 tier offering a total of 24 TB.

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.

When you add replicas, you can choose to host them in different Availability Zones. The benefit of distributing your replicas this way is that they're physically located in different data centers.

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

Now select to add any of these captured metrics:

- DocumentsProcessedCount
- SearchLatency
- SearchQueriesPerSecond
- SkillExecutionCount
- ThrottledSearchQueriesPercentage

For example, you could plot search latency against the percentage of throttled queries to see if the responses to queries are affected by throttling.

queries in kusto : <https://learn.microsoft.com/en-gb/training/modules/maintain-azure-cognitive-search-solution/06-monitor-azure-cognitive-search-solution>

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.

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.
- General document-Use this model to extract key-value pairs and tables in your documents.
- Layout- Use this model to extract text, tables, and structure information from forms. It can also recognize selection marks such as check boxes and radio buttons.

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

- Invoice-Use this model to extract key information from sales invoices in English and Spanish.
- Receipt-Use this model to extract data from printed and handwritten receipts.
- W-2 US tax declaration- Use this model to extract data from United States government's W-2 tax declaration form.
- ID Document-Use this model to extract data from United States driver's licenses and international passports.
- Business card-Use this model to extract names and contact details from business cards.
- Health insurance card-

It's built on the lower level Azure AI Services, including Azure AI Vision.

```
from azure.core.credentials import AzureKeyCredential
from azure.ai.documentintelligence import DocumentIntelligenceClient
```

```

from azure.ai.documentintelligence.models import AnalyzeResult

endpoint = "<your-endpoint>"
key = "<your-key>"

docUrl = "<url-of-document-to-analyze>"

document_analysis_client = DocumentIntelligenceClient(endpoint=endpoint,
credential=AzureKeyCredential(key))

poller = document_analysis_client.begin_analyze_document_from_url(
"prebuilt-document", docUrl)
result: AnalyzeResult = poller.result()

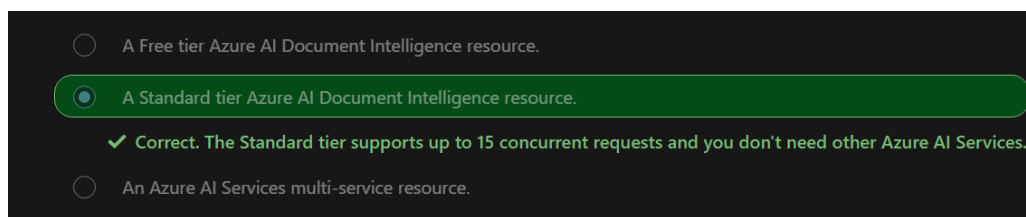
```

- **Custom template models.** A 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 template models support 9 different languages for handwritten text and a wide range of languages for printed text. If you have a few different variations of the form templates, train a model for each of the variations and then compose the models together into a single model. The service will invoke the model best suited to analyze the document.
- **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 Latin based languages like German, French, Italian, Spanish, and Dutch. Try using the custom neural model first if your scenario is addressed by the model.

A composed model is one that consists of multiple custom models.

If you're using the Standard pricing tier, you can add up to 100 custom models into a single composed model. If you're using the Free pricing tier, you can only add up to 5 custom models.

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



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.

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.

FOR READ :

For multi-page PDF or TIFF files, you can use the `pages` parameter in your request to fix a page range for the analysis.

The read model is ideal if you want to extract words and lines from documents with no fixed or predictable structure.

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

Word documents are not supported by Azure AI Document Intelligence but PDF documents are supported.

Azure AI Document Intelligence is designed to analyze scanned and photographed paper documents, not documents that are already in a digital format so you should consider using another technology to extract the data in Word documents.

Azure Document Intelligence is composed of the following services:

- **Document analysis models:** which take an input of JPEG, PNG, PDF, and TIFF files and return a JSON file with the location of text in bounding boxes, text content, tables, selection marks (also known as checkboxes or radio buttons), and document structure.
- **Prebuilt models:** which detect and extract information from document images and return the extracted data in a structured JSON output. Azure Document Intelligence currently supports prebuilt models for several forms, including:
 - W-2 forms
 - Invoices
 - Receipts
 - ID documents
 - Business cards
- **Custom models:** custom models extract data from forms specific to your business. Custom models can be trained through the [Azure Document Intelligence Studio](#).

Azure Document Intelligence works on input documents that meet these requirements:

- Format must be JPG, PNG, BMP, PDF (text or scanned), or TIFF.
- The file size must be less than 500 MB for paid (S0) tier and 4 MB for free (F0) tier.
- Image dimensions must be between 50 × 50 pixels and 10000 × 10000 pixels.
- The total size of the training data set must be 500 pages or less.

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.

A composed model consists of multiple custom models. When you submit a form for analysis, Azure AI Document Intelligence categorizes it and selects the best custom model to use for the analysis. This categorization means you don't have to keep track of the correct custom model yourself and specify it in the request.

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.

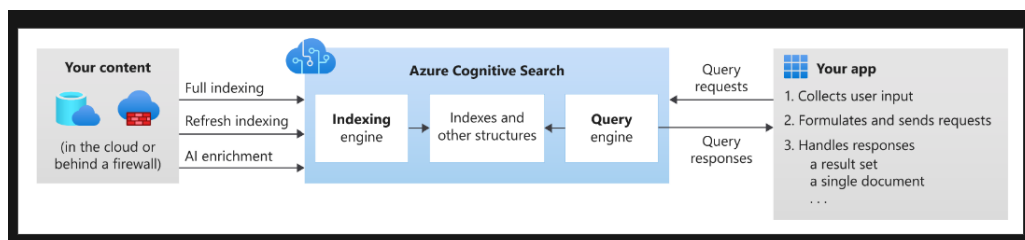
Create a composed model in code

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:

The docType property includes the model ID of the custom model that was used to analyze the document.

Only custom models that have been trained with labeled example forms can be added to a composed model.



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.

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.

2. You're troubleshooting your AI Search indexing process. You have a single custom skill that calls Azure AI Document Intelligence but requests are never received by your skill. Which of the following stages of the indexing process might be causing the problem? *

☐ Push to index.

☒ Output field mapping.

✗ Incorrect. The output field mapping stage happens after the skillset is complete.

☐ Document cracking.

✓ Correct. Since the document cracking stage happens before the skillset execution, it might prevent requests from reaching your custom skill.

Azure Open AI Service

Azure OpenAI includes several types of model:

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

The Completions playground allows you to make calls to your deployed models through a text-in, text-out interface and to adjust parameters.

Completions 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.

The Chat playground is based on a conversation-in, message-out interface.

The Chat playground, like the Completions playground, also includes the Temperature parameter. 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.
- **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.
- **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.

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.

`Completion` is available for all `gpt-3` generation models, while `ChatCompletion` is the only supported option for `gpt-4` models and is the preferred endpoint when using the `gpt-35-turbo` model. The lab in this module uses `gpt-35-turbo` with the `ChatCompletion` 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

openai sdk

The necessary parameters are `endpoint`, `key`, and the name of your deployment, which is called the `engine` when sending your prompt to the model.

The response object contains several values, such as `total_tokens` and `finish_reason`. The completion from the response object will be similar to the following completion:

1. What resource values are required to make requests to your Azure OpenAI resource? *

☐ Chat, Embedding, and Completion

☒ Key, Endpoint, and Deployment name

✓ Each call to the Azure OpenAI service requires a key, endpoint, and deployment name.

☐ Summary, Deployment name, and Endpoint

2. What are the three available endpoints for interacting with a deployed Azure OpenAI model? *

☐ Completion, ChatCompletion, and Translation

☒ Completion, ChatCompletion, and Embeddings

✓ Completion, ChatCompletion, and Embeddings are the three available endpoints

☐ Deployment, Summary, and Similarity

3. What is the best available endpoint to model the next completion of a conversation in Azure OpenAI? *

☒ ChatCompletion

✓ The best model to use for the next response in a conversation is ChatCompletion

☐ Embeddings

prompt engineering

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

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. Providing supporting content allows the model to respond more completely, accurately, and be more likely to include the desired information.

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 are leading words for the model to build upon, and often help shape the response in the right direction.

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

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.

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.

Another technique for improved interaction is to divide complex prompts into multiple queries.

DALL-E

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

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

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

- **prompt**: The description of the image to be generated.
- **n**: The number of images to be generated.
- **size**: The resolution of the image(s) to be generated (`256x256`, `512x512`, or `1024x1024`).

The **result** element includes a collection of **url** elements, each of which references a PNG image file generated from the prompt. In this example, the file might look similar to the following image:

There is DALL-E playground for image generation

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 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.

Adding your data is done through the Azure OpenAI Studio, in the **Chat** playground

When adding your data, you can choose to upload your data files, use data in a blob storage account, or connect to an existing AI Search index.

Azure OpenAI on your data supports `.md`, `.txt`, `.html`, `.pdf`, and Microsoft Word or PowerPoint files. If any of these files contain graphics or images, the response quality depends on how well text can be extracted from the visual content.

When uploading data or connecting to files in a storage account, it's recommended to use the Azure OpenAI Studio to create the search resource and index. Adding data this way allows the appropriate chunking to happen when inserting into the index, yielding better responses. If you're using large text files or forms, you should use the available [data preparation script](#) to improve the AI model's accuracy.

Each call to the model includes tokens for the system message, the user prompt, conversation history, retrieved search documents, internal prompts, and the model's response.

While there's no token limit for the system message, when using your own data the system message gets truncated if it exceeds 200 tokens. The response from the model is also limited when using your own data is 1500 tokens.

active

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.

The request will also need to include the `Content-Type` and `api-key`.

Responsible genai

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.

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.
5. *Identify* potential harms that are relevant to your planned solution.
6. *Measure* the presence of these harms in the outputs generated by your solution.
7. *Mitigate* the harms at multiple layers in your solution to minimize their presence and impact, and ensure transparent communication about potential risks to users.
8. *Operate* the solution responsibly by defining and following a deployment and operational readiness plan.
9. *Identify* potential harms that are relevant to your planned solution.
10. *Measure* the presence of these harms in the outputs generated by your solution.
11. *Mitigate* the harms at multiple layers in your solution to minimize their presence and impact, and ensure transparent communication about potential risks to users.
12. *Operate* the solution responsibly by defining and following a deployment and operational readiness plan.

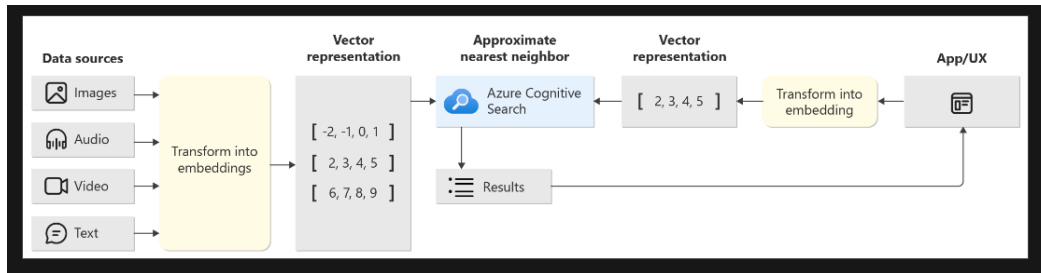
Semantic ranking is a capability within Azure AI Search that aims to improve the ranking of search results.

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 cannot be enabled or disabled on a per-index basis.

Before enabling semantic ranking, you must have an Azure AI Search service with at least one index.



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.

You need to encode your Azure AI Search query by sending it to an embedded model. The response is then passed to a search engine to complete a search over the vector fields.

In order for your query to work, you need do the following tasks:

You check if your search has vector fields by running an empty search, the result includes a vector field with a number array.

You can also look for a field named **vectorSearch** of with the type **Collection(Edm.single)**. This has an algorithm configuration and an attribute of 'dimension'.

You can only query a vector field with a query vector. Your end-users provide a text query string, which your application converts into a vector by using the embedding library you used for to create the source document embeddings.

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. For example, if two texts are similar, then their representations should also be similar.