Introduction to AI concepts

Key points to understand about generative AI include:

- Generative AI is a branch of AI that enables software applications to generate new content; often natural language dialogs, but also images, video, code, and other formats.
- The ability to generate content is based on a language model, which has been trained with huge volumes of data - often documents from the Internet or other public sources of information.
- Generative AI models encapsulate semantic relationships between language elements (that's a fancy way of saying that the models "know" how words relate to one another), and that's what enables them to generate a meaningful sequence of text.
- There are large language models (LLMs) and small language models (SLMs) - the difference is based on the volume of data and the number of variables in the model. LLMs are very powerful and generalize well, but can be more costly to train and use. SLMs tend to work well in scenarios that are more focused on specific topic areas, and usually cost less.

Generative AI scenarios

Common uses of generative AI include:

- Implementing chatbots and AI agents that assist human users.
- Creating new documents or other content (often as a starting point for further iterative development)
- Automated translation of text between languages.
- Summarizing or explaining complex documents.

Key points to understand about computer vision include:

- Computer vision is accomplished by using large numbers of images to train a model.
- Image classification is a form of computer vision in which a model is trained with images that are labeled with the main subject of the image (in other words, what it's an image of) so that it can analyze unlabeled images and predict the most appropriate label - identifying the subject of the image.

- Object detection is a form of computer vision in which the model is trained to identify the location of specific objects in an image.
- There are more advanced forms of computer vision - for example, semantic segmentation is an advanced form of object detection where, rather than indicate an object's location by drawing a box around it, the model can identify the individual pixels in the image that belong to a particular object.
- You can combine computer vision and language models to create a multi-modal model that combines computer vision and generative AI capabilities.

Computer vision scenarios

Common uses of computer vision include:

- Auto-captioning or tag-generation for photographs.
- Visual search.
- Monitoring stock levels or identifying items for checkout in retail scenarios.
- Security video monitoring.
- Authentication through facial recognition.
- Robotics and self-driving vehicles.

Key points to understand about speech include:

- Speech recognition is the ability of AI to "hear" and interpret speech. Usually this capability takes the form of speech-to-text (where the audio signal for the speech is transcribed into text).
- Speech synthesis is the ability of AI to vocalize words as spoken language. Usually this capability takes the form of text-to-speech in which information in text format is converted into an audible signal.
- AI speech technology is evolving rapidly to handle challenges like ignoring background noise, detecting interruptions, and generating increasingly expressive and human-like voices.

AI speech scenarios

Common uses of AI speech technologies include:

- Personal AI assistants in phones, computers, or household devices with which you interact by talking.
- Automated transcription of calls or meetings.
- Automating audio descriptions of video or text.
- Automated speech translation between languages.

Key points to understand about natural language processing (NLP) include:

- NLP capabilities are based on models that are trained to do particular types of text analysis.
- While many natural language processing scenarios are handled by generative AI models today, there are many common text analytics use cases where simpler NLP language models can be more cost-effective.

Common NLP tasks include:

- Entity extraction - identifying mentions of entities like people, places, organizations in a document
- Text classification - assigning document to a specific category.
- Sentiment analysis - determining whether a body of text is positive, negative, or neutral and inferring opinions.
- Language detection - identifying the language in which text is written.

Note

In this module, we've used the term natural language processing (NLP) to describe AI capabilities that derive meaning from "ordinary" human language. You might also see this area of AI referred to as natural language understanding (NLU).

Natural language processing scenarios

Common uses of NLP technologies include:

- Analyzing document or transcripts of calls and meetings to determine key subjects and identify specific mentions of people, places, organizations, products, or other entities.

- Analyzing social media posts, product reviews, or articles to evaluate sentiment and opinion.

- Implementing chatbots that can answer frequently asked questions or orchestrate predictable conversational dialogs that don't require the complexity of generative AI.

Key points to understand about using AI to extract data and insights include:

- The basis for most document analysis solutions is a computer vision technology called optical character recognition (OCR).

- While an OCR model can identify the location of text in an image, more advanced models can also interpret individual values in the document - and so extract specific fields.

- While most data extraction models have historically focused on extracting fields from text-based forms, more advanced models that can extract information from audio recording, images, and videos are becoming more readily available.

Data and insight extraction scenarios

Common uses of AI to extract data and insights include:

- Automated processing of forms and other documents in a business process - for example, processing an expense claim.

- Large-scale digitization of data from paper forms. For example, scanning and archiving census records.

- Indexing documents for search.

- Identifying key points and follow-up actions from meeting transcripts or recordings.

Key points to understand about responsible AI include:

- Fairness: AI models are trained using data, which is generally sourced and selected by humans. There's substantial risk that the data selection criteria, or the data itself reflects unconscious bias that may cause a model to produce discriminatory outputs. AI developers need to take care to minimize bias in training data and test AI systems for fairness.

- Reliability and safety: AI is based on probabilistic models, it is not infallible. AI-powered applications need to take this into account and mitigate risks accordingly.

- Privacy and security: Models are trained using data, which may include personal information. AI developers have a responsibility to ensure that the training data is kept secure, and that the trained models themselves can't be used to reveal private personal or organizational details.

- Inclusiveness: The potential of AI to improve lives and drive success should be open to everyone. AI developers should strive to ensure that their solutions don't exclude some users.

- Transparency: AI can sometimes seem like "magic", but it's important to make users aware of how the system works and any potential limitations it may have.

- Accountability: Ultimately, the people and organizations that develop and distribute AI solutions are accountable for their actions. It's important for organizations developing AI models and applications to define and apply a framework of governance to help ensure that they apply responsible AI principles to their work.

Responsible AI examples

- An AI-powered college admissions system should be tested to ensure it evaluates all applications fairly, taking into account relevant academic criteria but avoiding unfounded discrimination based on irrelevant demographic factors.

- An AI-powered robotic solution that uses computer vision to detect objects should avoid unintentional harm or damage. One way to accomplish this goal is to use probability values to determine "confidence" in object identification before interacting with physical objects, and avoid any action if the confidence level is below a specific threshold.

- A facial identification system used in an airport or other secure area should delete personal images that are used for temporary access as soon as they're no longer required. Additionally, safeguards should prevent the images being made accessible to operators or users who have no need to view them.

- A web-based chatbot that offers speech-based interaction should also generate text captions to avoid making the system unusable for users with a hearing impairment.

- A bank that uses an AI-based loan-approval application should disclose the use of AI, and describe features of the data on which it was trained (without revealing confidential information).

Introduction to machine learning concepts:

Machine learning is in many ways the intersection of two disciplines - data science and software engineering. The goal of machine learning is to use data to create a predictive model that can be incorporated into a software application or service. To achieve this goal requires collaboration between data scientists who explore and prepare the data before using it to *train* a machine learning model, and software developers who integrate the models into applications where they're used to predict new data values (a process known as *inferencing*).

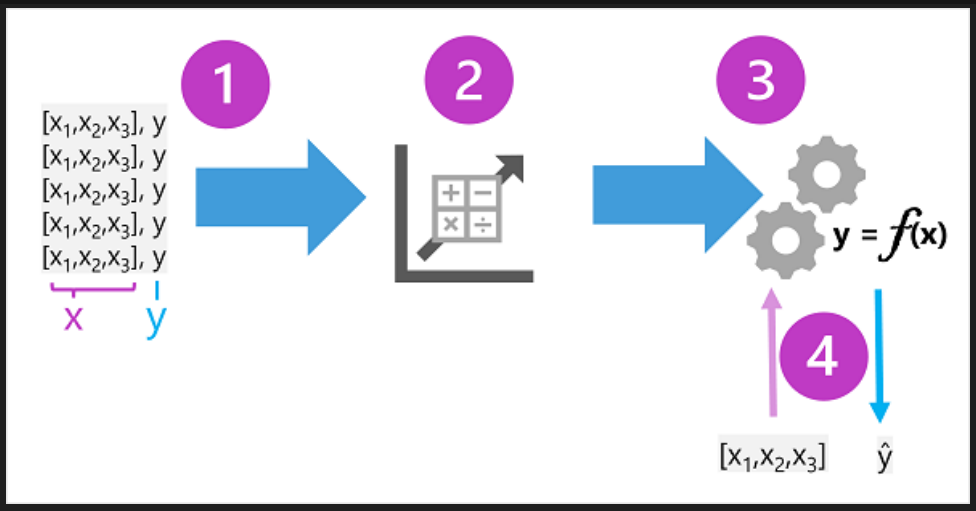
Machine learning has its origins in statistics and mathematical modeling of data. The fundamental idea of machine learning is to use data from past observations to predict unknown outcomes or values. For example:

- The proprietor of an ice cream store might use an app that combines historical sales and weather records to predict how many ice creams they're likely to sell on a given day, based on the weather forecast.
- A doctor might use clinical data from past patients to run automated tests that predict whether a new patient is at risk from diabetes based on factors like weight, blood glucose level, and other measurements.
- A researcher in the Antarctic might use past observations to automate the identification of different penguin species (such as *Adelie*, *Gentoo*, or *Chinstrap*) based on measurements of a bird's flippers, bill, and other physical attributes.

Machine learning models

Because machine learning is based on mathematics and statistics, it's common to think about machine learning models in mathematical terms. Fundamentally, a machine learning model is a software application that encapsulates a *function* to calculate an output value based on one or more input values. The process of defining that function is known as *training*. After the function has been defined, you can use it to predict new values in a process called *inferencing*.

Let's explore the steps involved in training and inferencing.

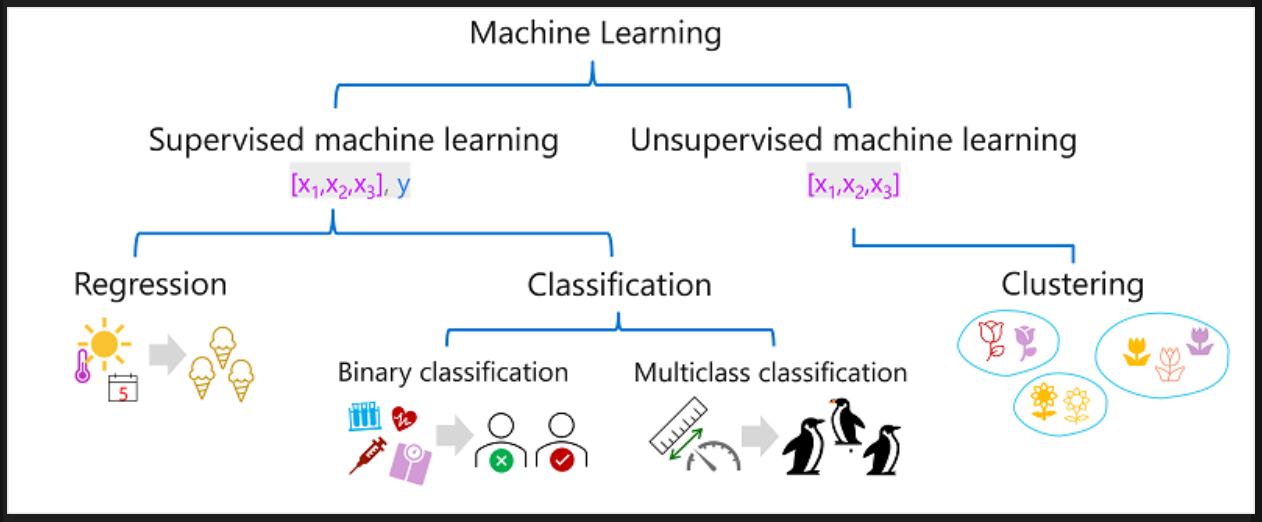


1. The training data consists of past observations. In most cases, the observations include the observed attributes or *features* of the thing being observed, and the known value of the thing you want to train a model to predict (known as the *label*).
   In mathematical terms, you'll often see the features referred to using the shorthand variable name *x*, and the label referred to as *y*. Usually, an observation consists of multiple feature values, so *x* is actually a *vector* (an array with multiple values), like this: *[x1,x2,x3,...]*.
   To make this clearer, let's consider the examples described previously:
   - In the ice cream sales scenario, our goal is to train a model that can predict the number of ice cream sales based on the weather. The weather measurements for the day (temperature, rainfall, windspeed, and so on) would be the *features* (*x*), and the number of ice creams sold on each day would be the *label* (*y*).
   - In the medical scenario, the goal is to predict whether or not a patient is at risk of diabetes based on their clinical measurements. The patient's measurements (weight, blood glucose level, and so on) are the *features* (*x*), and the likelihood of diabetes (for example, *1* for at risk, *0* for not at risk) is the *label* (*y*).
   - In the Antarctic research scenario, we want to predict the species of a penguin based on its physical attributes. The key measurements of the penguin (length of its flippers, width of its bill, and so on) are the *features* (*x*), and the species (for example, *0* for Adelie, *1* for Gentoo, or *2* for Chinstrap) is the *label* (*y*).
2. An *algorithm* is applied to the data to try to determine a relationship between the features and the label, and generalize that relationship as a calculation that can be performed on *x* to calculate *y*. The specific algorithm used depends on the kind of predictive problem you're trying to solve (more

about this later), but the basic principle is to try to *fit* the data to a function in which the values of the features can be used to calculate the label.
3. The result of the algorithm is a *model* that encapsulates the calculation derived by the algorithm as a *function* - let's call it *f*. In mathematical notation:
   $y = f(x)$
4. Now that the *training* phase is complete, the trained model can be used for *inferencing*. The model is essentially a software program that encapsulates the function produced by the training process. You can input a set of feature values, and receive as an output a prediction of the corresponding label. Because the output from the model is a prediction that was calculated by the function, and not an observed value, you'll often see the output from the function shown as $\hat{y}$ (which is rather delightfully verbalized as "y-hat").

Types of machine learning model

There are multiple types of machine learning, and you must apply the appropriate type depending on what you're trying to predict. A breakdown of common types of machine learning is shown in the following diagram.



Supervised machine learning

Supervised machine learning is a general term for machine learning algorithms in which the training data includes both *feature* values and known *label* values. Supervised machine learning is used to train models by determining a relationship between the features and labels in past observations, so that unknown labels can be predicted for features in future cases.

Regression

Regression is a form of supervised machine learning in which the label predicted by the model is a numeric value. For example:

- The number of ice creams sold on a given day, based on the temperature, rainfall, and windspeed.
- The selling price of a property based on its size in square feet, the number of bedrooms it contains, and socio-economic metrics for its location.
- The fuel efficiency (in miles-per-gallon) of a car based on its engine size, weight, width, height, and length.

Classification

Classification is a form of supervised machine learning in which the label represents a categorization, or *class*. There are two common classification scenarios.

Binary classification

In *binary classification*, the label determines whether the observed item *is* (or *isn't*) an instance of a specific class. Or put another way, binary classification models predict one of two mutually exclusive outcomes. For example:

- Whether a patient is at risk for diabetes based on clinical metrics like weight, age, blood glucose level, and so on.
- Whether a bank customer will default on a loan based on income, credit history, age, and other factors.
- Whether a mailing list customer will respond positively to a marketing offer based on demographic attributes and past purchases.

In all of these examples, the model predicts a binary *true*/*false* or *positive/negative* prediction for a single possible class.

Multiclass classification

Multiclass classification extends binary classification to predict a label that represents one of multiple possible classes. For example,

- The species of a penguin (*Adelie*, *Gentoo*, or *Chinstrap*) based on its physical measurements.
- The genre of a movie (*comedy*, *horror*, *romance*, *adventure*, or *science fiction*) based on its cast, director, and budget.

In most scenarios that involve a known set of multiple classes, multiclass classification is used to predict mutually exclusive labels. For example, a penguin can't be both a *Gentoo* and an *Adelie*. However, there are also some algorithms that you can use to train *multilabel* classification models, in which there may be more than one valid label for a single observation. For example, a movie could potentially be categorized as both *science fiction* and *comedy*.

Unsupervised machine learning

Unsupervised machine learning involves training models using data that consists only of *feature* values without any known labels. Unsupervised machine learning algorithms determine relationships between the features of the observations in the training data.

Clustering

The most common form of unsupervised machine learning is *clustering*. A clustering algorithm identifies similarities between observations based on their features, and groups them into discrete clusters. For example:

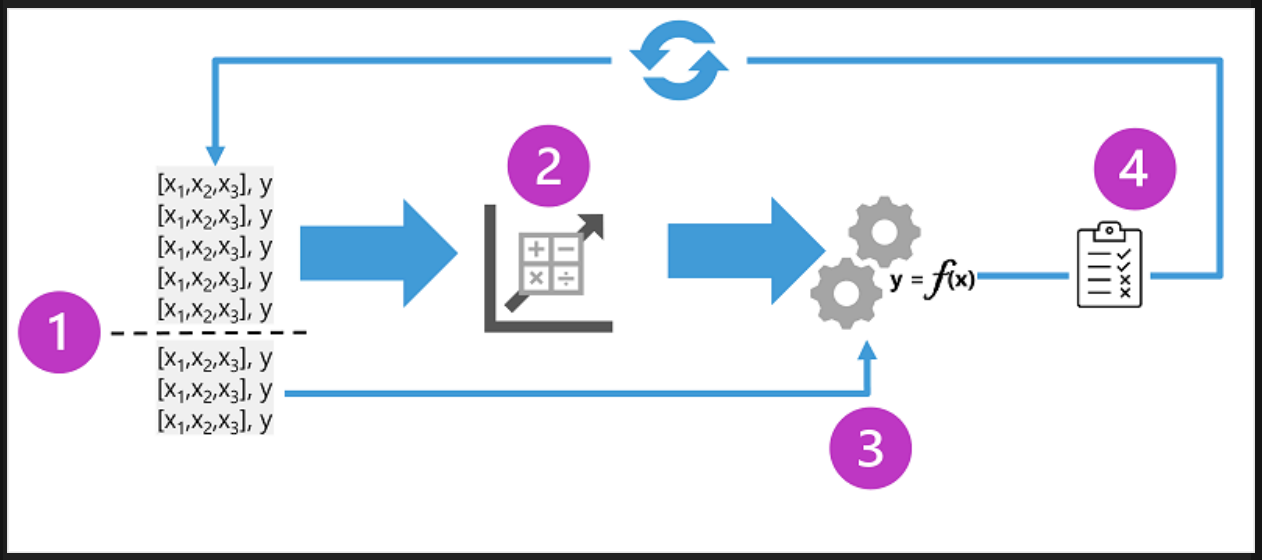
- Group similar flowers based on their size, number of leaves, and number of petals.
- Identify groups of similar customers based on demographic attributes and purchasing behavior.

In some ways, clustering is similar to multiclass classification; in that it categorizes observations into discrete groups. The difference is that when using classification, you already know the classes to which the observations in the training data belong; so the algorithm works by determining the relationship between the features and the known classification label. In clustering, there's no previously known cluster label and the algorithm groups the data observations based purely on similarity of features.

In some cases, clustering is used to determine the set of classes that exist before training a classification model. For example, you might use clustering to segment your customers into groups, and then analyze those groups to identify and categorize different classes of customer (*high value - low volume*, *frequent small purchaser*, and so on). You could then use your categorizations to label the observations in your clustering results and use the labeled data to train a classification model that predicts to which customer category a new customer might belong.

Regression

Regression models are trained to predict numeric label values based on training data that includes both features and known labels. The process for training a regression model (or indeed, any supervised machine learning model) involves multiple iterations in which you use an appropriate algorithm (usually with some parameterized settings) to train a model, evaluate the model's predictive performance, and refine the model by repeating the training process with different algorithms and parameters until you achieve an acceptable level of predictive accuracy.



The diagram shows four key elements of the training process for supervised machine learning models:

1. Split the training data (randomly) to create a dataset with which to train the model while holding back a subset of the data that you'll use to validate the trained model.
2. Use an algorithm to fit the training data to a model. In the case of a regression model, use a regression algorithm such as *linear regression*.
3. Use the validation data you held back to test the model by predicting labels for the features.
4. Compare the known *actual* labels in the validation dataset to the labels that the model predicted. Then aggregate the differences between the *predicted* and *actual* label values to calculate a metric that indicates how accurately the model predicted for the validation data.

After each train, validate, and evaluate iteration, you can repeat the process with different algorithms and parameters until an acceptable evaluation metric is achieved.

Example - regression

Let's explore regression with a simplified example in which we'll train a model to predict a numeric label (*y*) based on a single feature value (*x*). Most real scenarios involve multiple feature values, which adds some complexity; but the principle is the same.

For our example, let's stick with the ice cream sales scenario we discussed previously. For our feature, we'll consider the *temperature* (let's assume the value is the maximum temperature on a given day), and the label we want to train a model to predict is the number of ice creams sold that day. We'll start with some historic data that includes records of daily temperatures (*x*) and ice cream sales (*y*):

Temperature (x)	Ice cream sales (y)
51	1
52	0
67	14
65	14
70	23
69	20

72	23
75	26
73	22
81	30
78	26
83	36

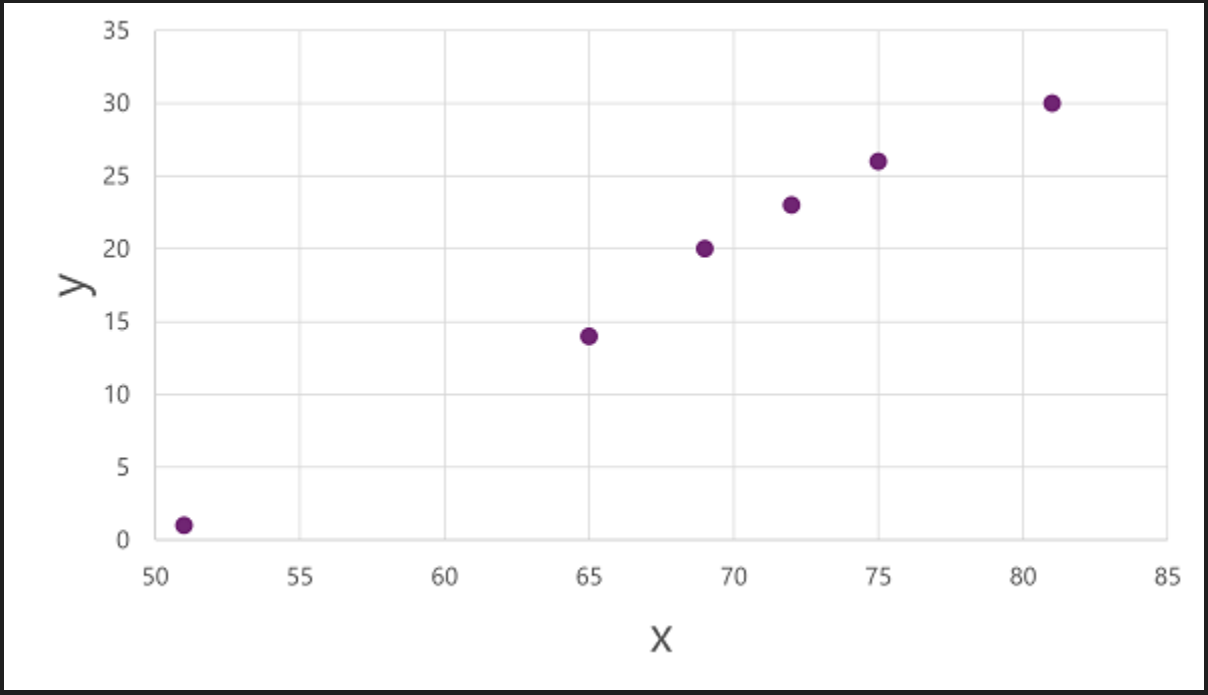
Training a regression model

We'll start by splitting the data and using a subset of it to train a model. Here's the training dataset:

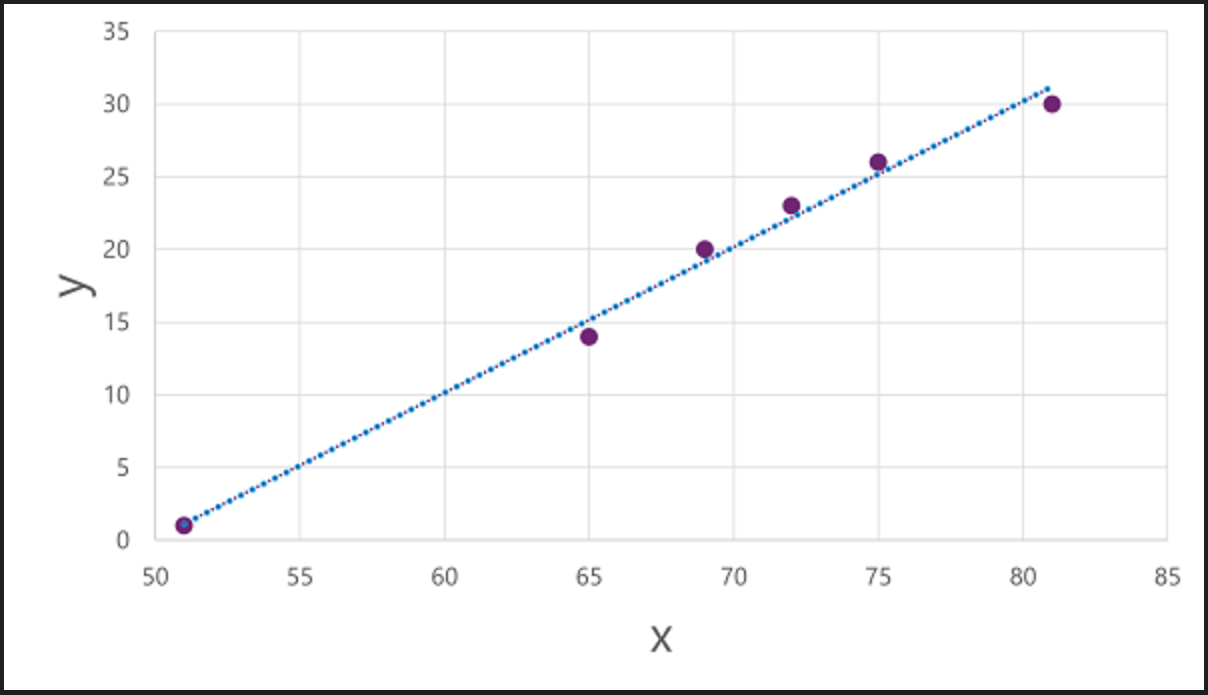
Temperature (x)	Ice cream sales (y)
51	1
65	14

69	20
72	23
75	26
81	30

To get an insight of how these *x* and *y* values might relate to one another, we can plot them as coordinates along two axes, like this:



Now we're ready to apply an algorithm to our training data and fit it to a function that applies an operation to *x* to calculate *y*. One such algorithm is *linear regression*, which works by deriving a function that produces a straight line through the intersections of the *x* and *y* values while minimizing the average distance between the line and the plotted points, like this:



The line is a visual representation of the function in which the slope of the line describes how to calculate the value of *y* for a given value of *x*. The line intercepts the *x* axis at 50, so when *x* is 50, *y* is 0. As you can see from the axis markers in the plot, the line slopes so that every increase of 5 along the *x* axis results in an increase of 5 up the *y* axis; so when *x* is 55, *y* is 5; when *x* is 60, *y* is 10, and so on. To calculate a value of *y* for a given value of *x*, the function simply subtracts 50; in other words, the function can be expressed like this:

f(x) = x-50

You can use this function to predict the number of ice creams sold on a day with any given temperature. For example, suppose the weather forecast tells us that tomorrow it will be 77 degrees. We can apply our model to calculate *77-50* and predict that we'll sell 27 ice creams tomorrow.

But just how accurate is our model?

Evaluating a regression model

To validate the model and evaluate how well it predicts, we held back some data for which we know the label (*y*) value. Here's the data we held back:

Temperature (x)	Ice cream sales (y)

52	0
67	14
70	23
73	22
78	26
83	36

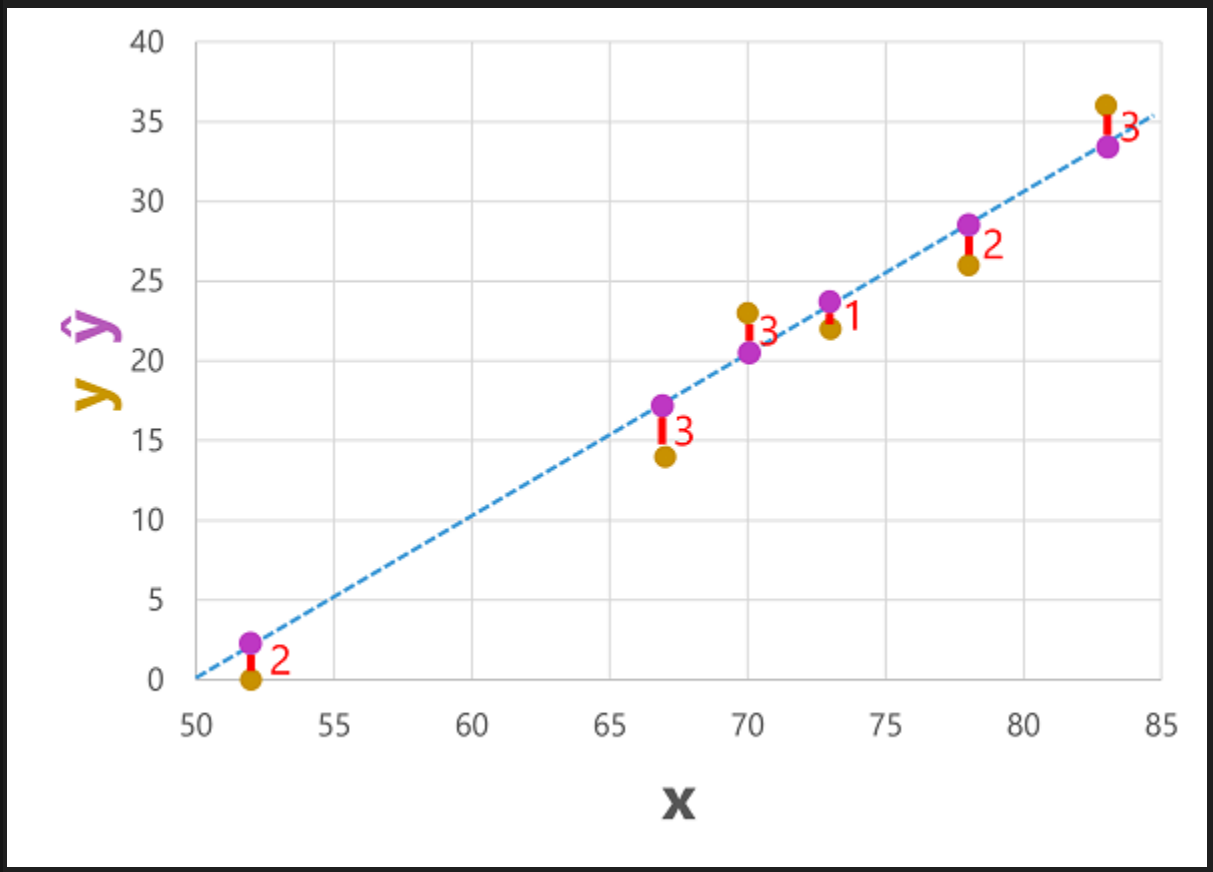
We can use the model to predict the label for each of the observations in this dataset based on the feature (*x*) value; and then compare the predicted label (*ŷ*) to the known actual label value (*y*).

Using the model we trained earlier, which encapsulates the function *f*(x) = x-50, results in the following predictions:

Temperature (x)	**Actual sales (y)**	**Predicted sales (ŷ)**
52	0	2
67	14	17

70	23	20
73	22	23
78	26	28
83	36	33

We can plot both the *predicted* and *actual* labels against the feature values like this:



The predicted labels are calculated by the model so they're on the function line, but there's some variance between the $\hat{y}$ values calculated by the function and the actual y

values from the validation dataset; which is indicated on the plot as a line between the $\hat{y}$ and y values that shows how far off the prediction was from the actual value.

Regression evaluation metrics

Based on the differences between the predicted and actual values, you can calculate some common metrics that are used to evaluate a regression model.

Mean Absolute Error (MAE)

The variance in this example indicates by how many ice creams each prediction was wrong. It doesn't matter if the prediction was *over* or *under* the actual value (so for example, -3 and +3 both indicate a variance of 3). This metric is known as the *absolute error* for each prediction, and can be summarized for the whole validation set as the mean absolute error (MAE).

In the ice cream example, the mean (average) of the absolute errors (2, 3, 3, 1, 2, and 3) is 2.33.

Mean Squared Error (MSE)

The mean absolute error metric takes all discrepancies between predicted and actual labels into account equally. However, it may be more desirable to have a model that is consistently wrong by a small amount than one that makes fewer, but larger errors. One way to produce a metric that "amplifies" larger errors by *squaring* the individual errors and calculating the mean of the squared values. This metric is known as the mean squared error (MSE).

In our ice cream example, the mean of the squared absolute values (which are 4, 9, 9, 1, 4, and 9) is 6.

Root Mean Squared Error (RMSE)

The mean squared error helps take the magnitude of errors into account, but because it *squares* the error values, the resulting metric no longer represents the quantity measured by the label. In other words, we can say that the MSE of our model is 6, but that doesn't measure its accuracy in terms of the number of ice creams that were mispredicted; 6 is just a numeric score that indicates the level of error in the validation predictions.

If we want to measure the error in terms of the number of ice creams, we need to calculate the *square root* of the MSE; which produces a metric called, unsurprisingly, Root Mean Squared Error. In this case $\sqrt{6}$, which is 2.45 (ice creams).

Coefficient of determination (R2)

All of the metrics so far compare the discrepancy between the predicted and actual values in order to evaluate the model. However, in reality, there's some natural random variance in the daily sales of ice cream that the model takes into account. In a linear regression model, the training algorithm fits a straight line that minimizes the mean variance between the function and the known label values. The coefficient of determination (more commonly referred to as R^2 or R-Squared) is a metric that measures the proportion of variance in the validation results that can be explained by the model, as opposed to some anomalous aspect of the validation data (for example, a day with a highly unusual number of ice creams sales because of a local festival).

The calculation for R^2 is more complex than for the previous metrics. It compares the sum of squared differences between predicted and actual labels with the sum of squared differences between the actual label values and the mean of actual label values, like this:

$$R^2 = 1 - \sum(y-\hat{y})^2 \div \sum(y-\bar{y})^2$$

Don't worry too much if that looks complicated; most machine learning tools can calculate the metric for you. The important point is that the result is a value between 0 and 1 that describes the proportion of variance explained by the model. In simple terms, the closer to 1 this value is, the better the model is fitting the validation data. In the case of the ice cream regression model, the R^2 calculated from the validation data is 0.95.

Iterative training

The metrics described above are commonly used to evaluate a regression model. In most real-world scenarios, a data scientist will use an iterative process to repeatedly train and evaluate a model, varying:

- Feature selection and preparation (choosing which features to include in the model, and calculations applied to them to help ensure a better fit).
- Algorithm selection (We explored linear regression in the previous example, but there are many other regression algorithms)
- Algorithm parameters (numeric settings to control algorithm behavior, more accurately called *hyperparameters* to differentiate them from the *x* and *y* parameters).

After multiple iterations, the model that results in the best evaluation metric that's acceptable for the specific scenario is selected.

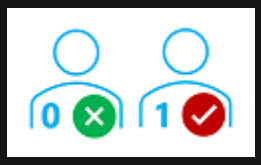
Binary classification

Classification, like regression, is a *supervised* machine learning technique; and therefore follows the same iterative process of training, validating, and evaluating models. Instead of calculating numeric values like a regression model, the algorithms used to train classification models calculate *probability* values for class assignment and the evaluation metrics used to assess model performance compare the predicted classes to the actual classes.

Binary classification algorithms are used to train a model that predicts one of two possible labels for a single class. Essentially, predicting *true* or *false*. In most real scenarios, the data observations used to train and validate the model consist of multiple feature (*x*) values and a *y* value that is either 1 or 0.

Example - binary classification

To understand how binary classification works, let's look at a simplified example that uses a single feature (*x*) to predict whether the label *y* is 1 or 0. In this example, we'll use the blood glucose level of a patient to predict whether or not the patient has diabetes. Here's the data with which we'll train the model:



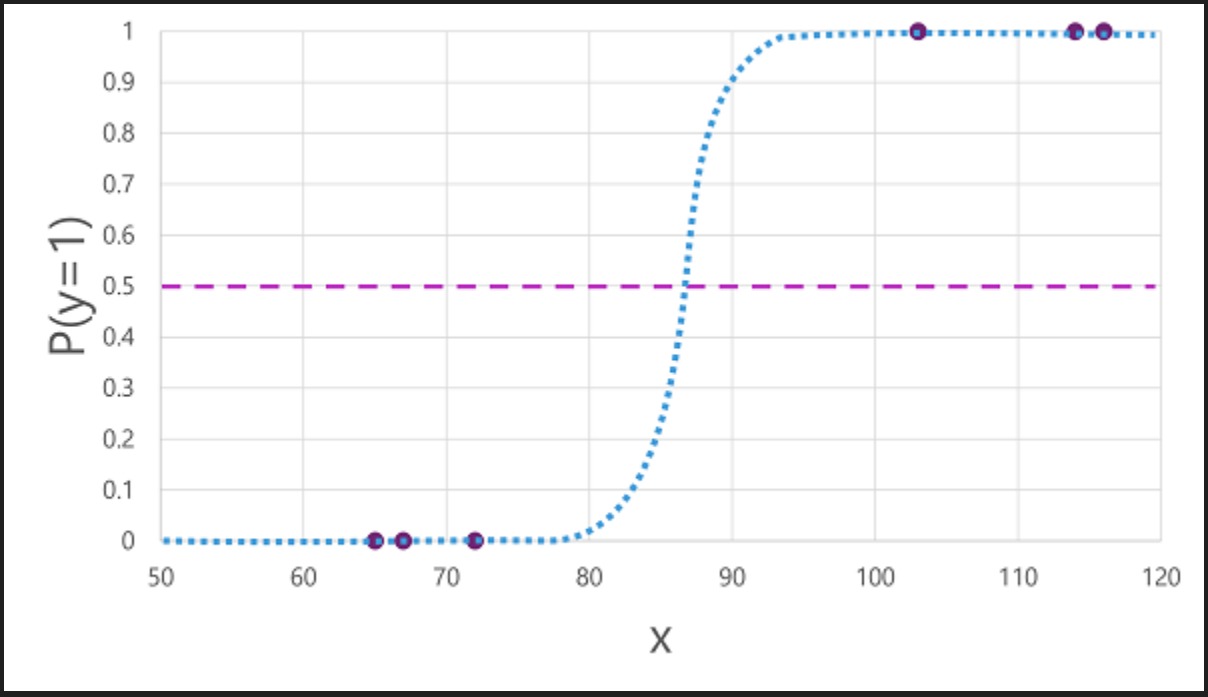
Blood glucose (x)	Diabetic? (y)
67	0
103	1
114	1

72	0
116	1
65	0

Training a binary classification model

To train the model, we'll use an algorithm to fit the training data to a function that calculates the *probability* of the class label being *true* (in other words, that the patient has diabetes). Probability is measured as a value between 0.0 and 1.0, such that the *total* probability for *all* possible classes is 1.0. So for example, if the probability of a patient having diabetes is 0.7, then there's a corresponding probability of 0.3 that the patient isn't diabetic.

There are many algorithms that can be used for binary classification, such as *logistic regression*, which derives a *sigmoid* (S-shaped) function with values between 0.0 and 1.0, like this:



Note

Despite its name, in machine learning *logistic regression* is used for classification, not regression. The important point is the *logistic* nature of the function it produces, which describes an S-shaped curve between a lower and upper value (0.0 and 1.0 when used for binary classification).

The function produced by the algorithm describes the probability of *y* being true (*y*=1) for a given value of *x*. Mathematically, you can express the function like this:

$f(x) = P(y=1 \mid x)$

For three of the six observations in the training data, we know that *y* is definitely *true*, so the probability for those observations that *y*=1 is 1.0 and for the other three, we know that *y* is definitely *false*, so the probability that *y*=1 is 0.0. The S-shaped curve describes the probability distribution so that plotting a value of *x* on the line identifies the corresponding probability that *y* is 1.

The diagram also includes a horizontal line to indicate the *threshold* at which a model based on this function will predict *true* (1) or *false* (0). The threshold lies at the mid-point for *y* (*P(y) = 0.5*). For any values at this point or above, the model will predict *true* (1); while for any values below this point it will predict *false* (0). For example, for a patient with a blood glucose level of 90, the function would result in a probability value of 0.9. Since 0.9 is higher than the threshold of 0.5, the model would predict *true* (1) - in other words, the patient is predicted to have diabetes.

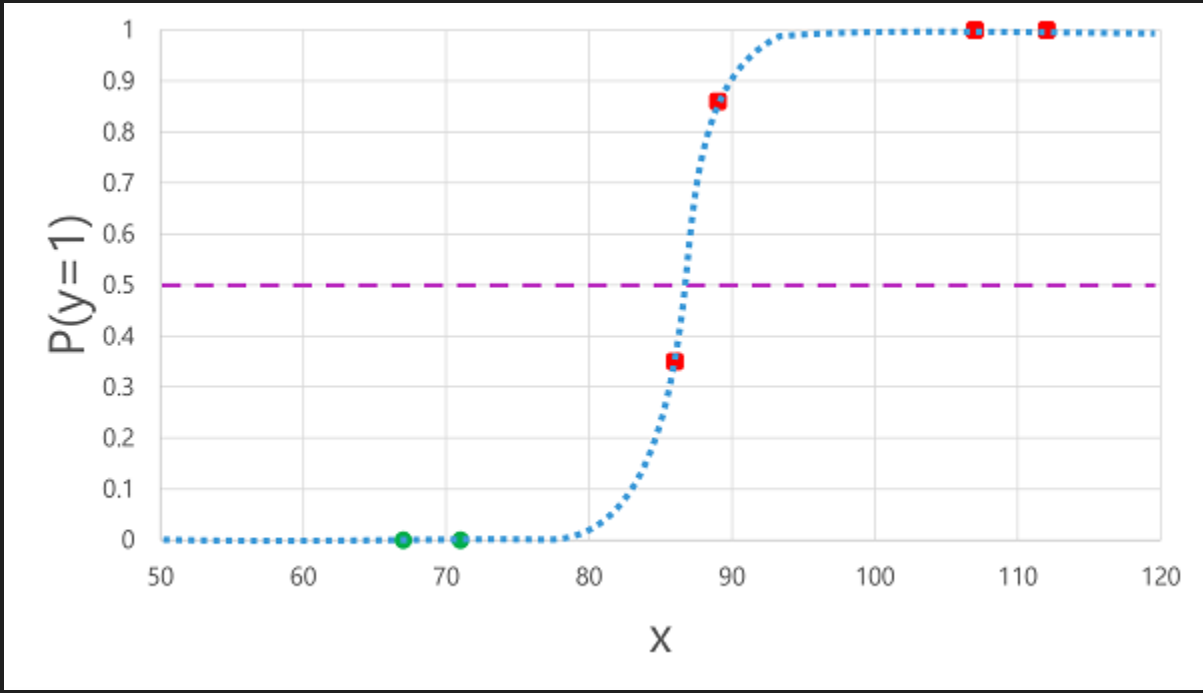
Evaluating a binary classification model

As with regression, when training a binary classification model you hold back a random subset of data with which to validate the trained model. Let's assume we held back the following data to validate our diabetes classifier:

Blood glucose (x)	Diabetic? (y)
66	0
107	1

112	1
71	0
87	1
89	1

Applying the logistic function we derived previously to the *x* values results in the following plot.

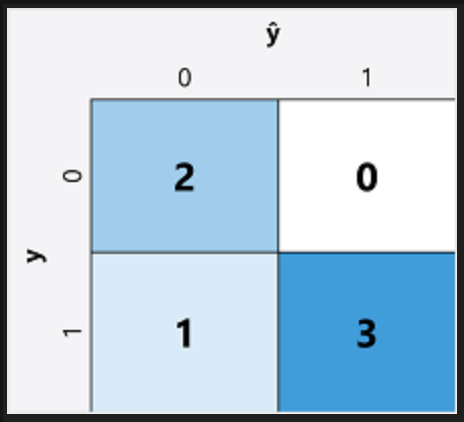


Based on whether the probability calculated by the function is above or below the threshold, the model generates a predicted label of 1 or 0 for each observation. We can then compare the *predicted* class labels ($\hat{y}$) to the *actual* class labels (y), as shown here:

Blood glucose (x)	Actual diabetes diagnosis (y)	Predicted diabetes diagnosis (ŷ)
66	0	0
107	1	1
112	1	1
71	0	0
87	1	0
89	1	1

Binary classification evaluation metrics

The first step in calculating evaluation metrics for a binary classification model is usually to create a matrix of the number of correct and incorrect predictions for each possible class label:



This visualization is called a *confusion matrix*, and it shows the prediction totals where:

- ŷ=0 and y=0: *True negatives* (TN)
- ŷ=1 and y=0: *False positives* (FP)
- ŷ=0 and y=1: *False negatives* (FN)
- ŷ=1 and y=1: *True positives* (TP)

The arrangement of the confusion matrix is such that correct (*true*) predictions are shown in a diagonal line from top-left to bottom-right. Often, color-intensity is used to indicate the number of predictions in each cell, so a quick glance at a model that predicts well should reveal a deeply shaded diagonal trend.

Accuracy

The simplest metric you can calculate from the confusion matrix is *accuracy* - the proportion of predictions that the model got right. Accuracy is calculated as:

(TN+TP) ÷ (TN+FN+FP+TP)

In the case of our diabetes example, the calculation is:

(2+3) ÷ (2+1+0+3)

= 5 ÷ 6

= 0.83

So for our validation data, the diabetes classification model produced correct predictions 83% of the time.

Accuracy might initially seem like a good metric to evaluate a model, but consider this. Suppose 11% of the population has diabetes. You could create a model that always predicts 0, and it would achieve an accuracy of 89%, even though it makes no real attempt to differentiate between patients by evaluating their features. What we really

need is a deeper understanding of how the model performs at predicting 1 for positive cases and 0 for negative cases.

Recall

Recall is a metric that measures the proportion of positive cases that the model identified correctly. In other words, compared to the number of patients who *have* diabetes, how many did the model *predict* to have diabetes?

The formula for recall is:

TP ÷ (TP+FN)

For our diabetes example:

3 ÷ (3+1)

= 3 ÷ 4

= 0.75

So our model correctly identified 75% of patients who have diabetes as having diabetes.

Precision

Precision is a similar metric to recall, but measures the proportion of predicted positive cases where the true label is actually positive. In other words, what proportion of the patients *predicted* by the model to have diabetes actually *have* diabetes?

The formula for precision is:

TP ÷ (TP+FP)

For our diabetes example:

3 ÷ (3+0)

= 3 ÷ 3

= 1.0

So 100% of the patients predicted by our model to have diabetes do in fact have diabetes.

F1-score

F1-score is an overall metric that combines recall and precision. The formula for F1-score is:

(2 x Precision x Recall) ÷ (Precision + Recall)

For our diabetes example:

(2 x 1.0 x 0.75) ÷ (1.0 + 0.75)

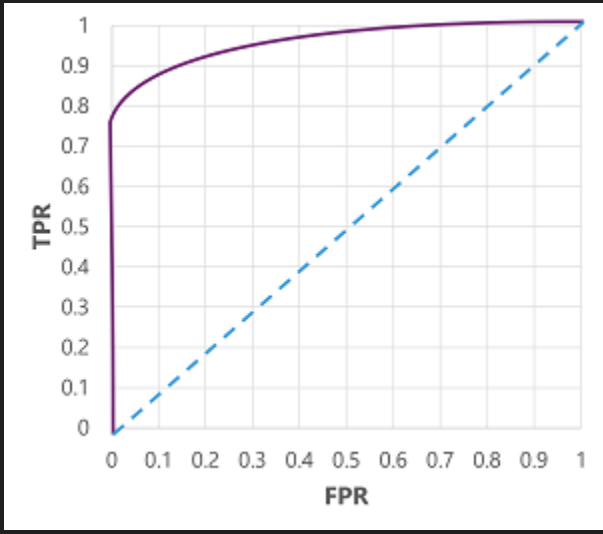
= 1.5 ÷ 1.75

= 0.86

Area Under the Curve (AUC)

Another name for recall is the *true positive rate* (TPR), and there's an equivalent metric called the *false positive rate* (FPR) that is calculated as FP÷(FP+TN). We already know that the TPR for our model when using a threshold of 0.5 is 0.75, and we can use the formula for FPR to calculate a value of 0÷2 = 0.

Of course, if we were to change the threshold above which the model predicts *true* (1), it would affect the number of positive and negative predictions; and therefore change the TPR and FPR metrics. These metrics are often used to evaluate a model by plotting a *received operator characteristic* (ROC) curve that compares the TPR and FPR for every possible threshold value between 0.0 and 1.0:



The ROC curve for a perfect model would go straight up the TPR axis on the left and then across the FPR axis at the top. Since the plot area for the curve measures 1x1, the area under this perfect curve would be 1.0 (meaning that the model is correct 100% of the time). In contrast, a diagonal line from the bottom-left to the top-right represents the results that would be achieved by randomly guessing a binary label; producing an area

under the curve of 0.5. In other words, given two possible class labels, you could reasonably expect to guess correctly 50% of the time.

In the case of our diabetes model, the curve above is produced, and the area under the curve (AUC) metric is 0.875. Since the AUC is higher than 0.5, we can conclude the model performs better at predicting whether or not a patient has diabetes than randomly guessing.

Multiclass classification

Multiclass classification is used to predict to which of multiple possible classes an observation belongs. As a supervised machine learning technique, it follows the same iterative *train, validate, and evaluate* process as regression and binary classification in which a subset of the training data is held back to validate the trained model.

Example - multiclass classification

Multiclass classification algorithms are used to calculate probability values for multiple class labels, enabling a model to predict the *most probable* class for a given observation.

Let's explore an example in which we have some observations of penguins, in which the flipper length (*x*) of each penguin is recorded. For each observation, the data includes the penguin species (*y*), which is encoded as follows:

- 0: Adelie
- 1: Gentoo
- 2: Chinstrap

Note

As with previous examples in this module, a real scenario would include multiple feature (*x*) values. We'll use a single feature to keep things simple.

Flipper length (x)	Species (y)
167	0
172	0
225	2
197	1
189	1
232	2
158	0

Training a multiclass classification model

To train a multiclass classification model, we need to use an algorithm to fit the training data to a function that calculates a probability value for each possible class. There are two kinds of algorithm you can use to do this:

- One-vs-Rest (OvR) algorithms
- Multinomial algorithms

One-vs-Rest (OvR) algorithms

One-vs-Rest algorithms train a binary classification function for each class, each calculating the probability that the observation is an example of the target class. Each function calculates the probability of the observation being a specific class compared to *any* other class. For our penguin species classification model, the algorithm would essentially create three binary classification functions:

- $f_0(x) = P(y=0 \mid x)$
- $f_1(x) = P(y=1 \mid x)$
- $f_2(x) = P(y=2 \mid x)$

Each algorithm produces a sigmoid function that calculates a probability value between 0.0 and 1.0. A model trained using this kind of algorithm predicts the class for the function that produces the highest probability output.

Multinomial algorithms

As an alternative approach is to use a multinomial algorithm, which creates a single function that returns a multi-valued output. The output is a *vector* (an array of values) that contains the *probability distribution* for all possible classes - with a probability score for each class which when totaled add up to 1.0:

$f(x) = [P(y=0|x), P(y=1|x), P(y=2|x)]$

An example of this kind of function is a *softmax* function, which could produce an output like the following example:

[0.2, 0.3, 0.5]

The elements in the vector represent the probabilities for classes 0, 1, and 2 respectively; so in this case, the class with the highest probability is 2.

Regardless of which type of algorithm is used, the model uses the resulting function to determine the most probable class for a given set of features (*x*) and predicts the corresponding class label (*y*).

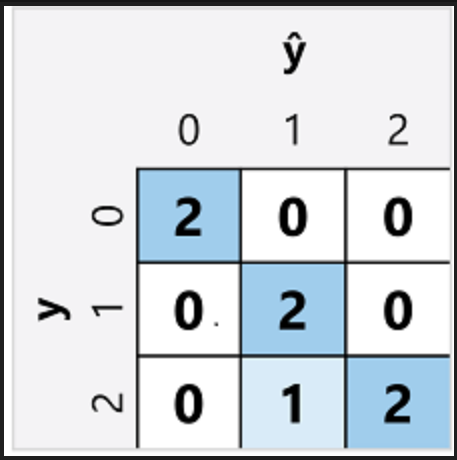
Evaluating a multiclass classification model

You can evaluate a multiclass classifier by calculating binary classification metrics for each individual class. Alternatively, you can calculate aggregate metrics that take all classes into account.

Let's assume that we've validated our multiclass classifier, and obtained the following results:

Flipper length (x)	Actual species (y)	Predicted species (ŷ)
165	0	0
171	0	0
205	2	1
195	1	1
183	1	1
221	2	2
214	2	2

The confusion matrix for a multiclass classifier is similar to that of a binary classifier, except that it shows the number of predictions for each combination of *predicted* ($\hat{y}$) and *actual* class labels (y):



From this confusion matrix, we can determine the metrics for each individual class as follows:

Class	True Positive	True Negative	False Positive	False Negative	Accuracy	Recall	Precision	F1-Score
0	2	5	0	0	1.0	1.0	1.0	1.0
1	2	4	1	0	0.86	1.0	0.67	0.8
2	2	4	0	1	0.86	0.67	1.0	0.8

To calculate the overall accuracy, recall, and precision metrics, you use the total of the *TP*, *TN*, *FP*, and *FN* metrics:

- Overall accuracy = (13+6)÷(13+6+1+1) = 0.90
- Overall recall = 6÷(6+1) = 0.86
- Overall precision = 6÷(6+1) = 0.86

The overall F1-score is calculated using the overall recall and precision metrics:

- Overall F1-score = (2x0.86x0.86)÷(0.86+0.86) = 0.86

Clustering

Clustering is a form of unsupervised machine learning in which observations are grouped into clusters based on similarities in their data values, or features. This kind of machine learning is considered unsupervised because it doesn't make use of previously known label values to train a model. In a clustering model, the label is the cluster to which the observation is assigned, based only on its features.

Example - clustering

For example, suppose a botanist observes a sample of flowers and records the number of leaves and petals on each flower:

There are no known *labels* in the dataset, just two *features*. The goal is not to identify the different types (species) of flower; just to group similar flowers together based on the number of leaves and petals.

Leaves (x_1)	**Petals** (x_2)
0	5
0	6
1	3

1	3
1	6
1	8
2	3
2	7
2	8

Training a clustering model

There are multiple algorithms you can use for clustering. One of the most commonly used algorithms is *K-Means* clustering, which consists of the following steps:

1. The feature (*x*) values are vectorized to define *n*-dimensional coordinates (where *n* is the number of features). In the flower example, we have two features: number of leaves (x_1) and number of petals (x_2). So, the feature vector has two coordinates that we can use to conceptually plot the data points in two-dimensional space ($[x_1,x_2]$)
2. You decide how many clusters you want to use to group the flowers - call this value *k*. For example, to create three clusters, you would use a *k* value of 3. Then *k* points are plotted at random coordinates. These points become the center points for each cluster, so they're called *centroids*.
3. Each data point (in this case a flower) is assigned to its nearest centroid.
4. Each centroid is moved to the center of the data points assigned to it based on the mean distance between the points.

5. After the centroid is moved, the data points may now be closer to a different centroid, so the data points are reassigned to clusters based on the new closest centroid.
6. The centroid movement and cluster reallocation steps are repeated until the clusters become stable or a predetermined maximum number of iterations is reached.

The following animation shows this process:

Evaluating a clustering model

Since there's no known label with which to compare the predicted cluster assignments, evaluation of a clustering model is based on how well the resulting clusters are separated from one another.

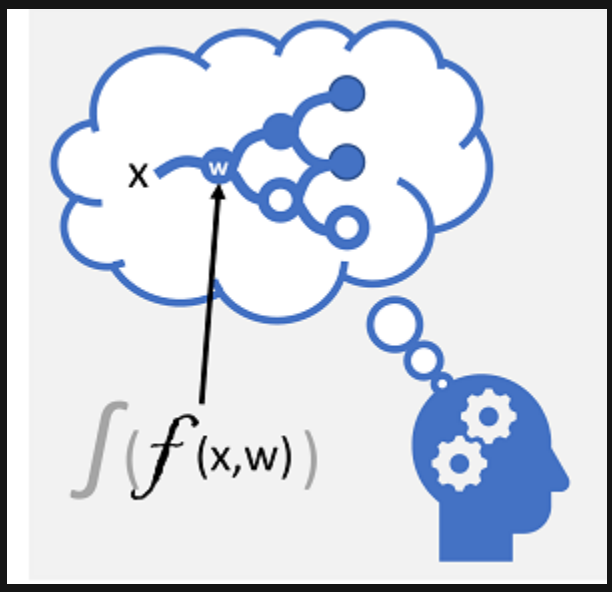
There are multiple metrics that you can use to evaluate cluster separation, including:

- Average distance to cluster center: How close, on average, each point in the cluster is to the centroid of the cluster.
- Average distance to other center: How close, on average, each point in the cluster is to the centroid of all other clusters.
- Maximum distance to cluster center: The furthest distance between a point in the cluster and its centroid.

- Silhouette: A value between -1 and 1 that summarizes the ratio of distance between points in the same cluster and points in different clusters (The closer to 1, the better the cluster separation).

Deep learning

Deep learning is an advanced form of machine learning that tries to emulate the way the human brain learns. The key to deep learning is the creation of an artificial *neural network* that simulates electrochemical activity in biological neurons by using mathematical functions, as shown here.

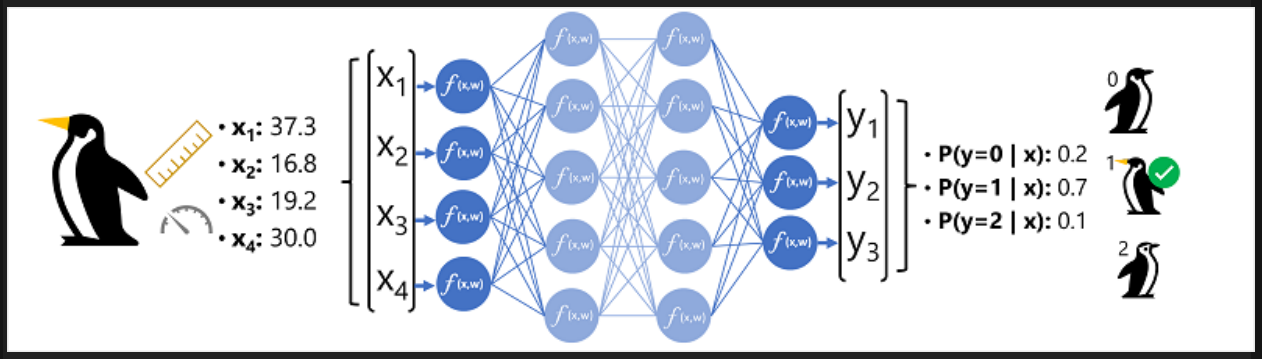
Biological neural network	Artificial neural network
	
Neurons fire in response to electrochemical stimuli. When fired, the signal is passed to connected neurons.	Each neuron is a function that operates on an input value (*x*) and a *weight* (*w*). The function is wrapped in an *activation* function that determines whether to pass the output on.

Artificial neural networks are made up of multiple *layers* of neurons - essentially defining a deeply nested function. This architecture is the reason the technique is referred to as *deep learning* and the models produced by it are often referred to as *deep neural networks* (DNNs). You can use deep neural networks for many kinds of machine learning problem, including regression and classification, as well as more specialized models for natural language processing and computer vision.

Just like other machine learning techniques discussed in this module, deep learning involves fitting training data to a function that can predict a label (*y*) based on the value of one or more features (*x*). The function (*f(x)*) is the outer layer of a nested function in which each layer of the neural network encapsulates functions that operate on *x* and the weight (*w*) values associated with them. The algorithm used to train the model involves iteratively feeding the feature values (*x*) in the training data forward through the layers to calculate output values for *ŷ*, validating the model to evaluate how far off the calculated *ŷ* values are from the known *y* values (which quantifies the level of error, or *loss*, in the model), and then modifying the weights (*w*) to reduce the loss. The trained model includes the final weight values that result in the most accurate predictions.

Example - Using deep learning for classification

To better understand how a deep neural network model works, let's explore an example in which a neural network is used to define a classification model for penguin species.



The feature data (*x*) consists of some measurements of a penguin. Specifically, the measurements are:

- The length of the penguin's bill.
- The depth of the penguin's bill.
- The length of the penguin's flippers.
- The penguin's weight.

In this case, *x* is a vector of four values, or mathematically, *x=[x1,x2,x3,x4]*.

The label we're trying to predict (*y*) is the species of the penguin, and that there are three possible species it could be:

- Adelie
- Gentoo
- Chinstrap

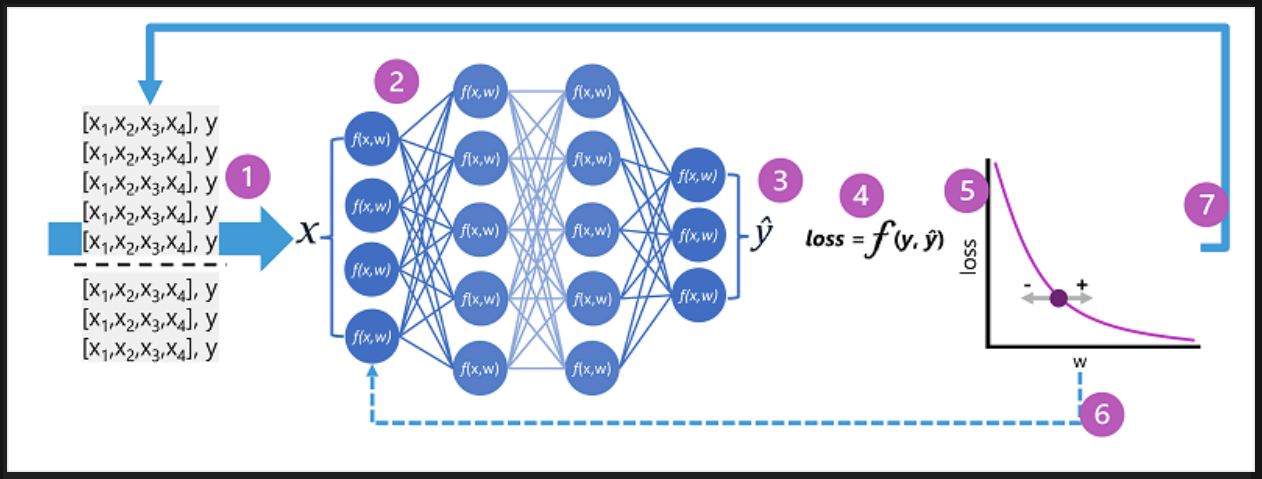
This is an example of a classification problem, in which the machine learning model must predict the most probable class to which an observation belongs. A classification model accomplishes this by predicting a label that consists of the probability for each class. In other words, y is a vector of three probability values; one for each of the possible classes: *[P(y=0|x), P(y=1|x), P(y=2|x)]*.

The process for inferencing a predicted penguin class using this network is:

1. The feature vector for a penguin observation is fed into the input layer of the neural network, which consists of a neuron for each *x* value. In this example, the following *x* vector is used as the input: *[37.3, 16.8, 19.2, 30.0]*
2. The functions for the first layer of neurons each calculate a weighted sum by combining the *x* value and *w* weight, and pass it to an activation function that determines if it meets the threshold to be passed on to the next layer.
3. Each neuron in a layer is connected to all of the neurons in the next layer (an architecture sometimes called a *fully connected network*) so the results of each layer are fed forward through the network until they reach the output layer.
4. The output layer produces a vector of values; in this case, using a *softmax* or similar function to calculate the probability distribution for the three possible classes of penguin. In this example, the output vector is: *[0.2, 0.7, 0.1]*
5. The elements of the vector represent the probabilities for classes 0, 1, and 2. The second value is the highest, so the model predicts that the species of the penguin is 1 (Gentoo).

How does a neural network learn?

The weights in a neural network are central to how it calculates predicted values for labels. During the training process, the model *learns* the weights that will result in the most accurate predictions. Let's explore the training process in a little more detail to understand how this learning takes place.



1. The training and validation datasets are defined, and the training features are fed into the input layer.
2. The neurons in each layer of the network apply their weights (which are initially assigned randomly) and feed the data through the network.
3. The output layer produces a vector containing the calculated values for $\hat{y}$. For example, an output for a penguin class prediction might be *[0.3. 0.1. 0.6]*.
4. A *loss function* is used to compare the predicted $\hat{y}$ values to the known y values and aggregate the difference (which is known as the *loss*). For example, if the known class for the case that returned the output in the previous step is *Chinstrap*, then the y value should be *[0.0, 0.0, 1.0]*. The absolute difference between this and the $\hat{y}$ vector is *[0.3, 0.1, 0.4]*. In reality, the loss function calculates the aggregate variance for multiple cases and summarizes it as a single *loss* value.
5. Since the entire network is essentially one large nested function, an optimization function can use differential calculus to evaluate the influence of each weight in the network on the loss, and determine how they could be adjusted (up or down) to reduce the amount of overall loss. The specific optimization technique can vary, but usually involves a *gradient descent* approach in which each weight is increased or decreased to minimize the loss.
6. The changes to the weights are *backpropagated* to the layers in the network, replacing the previously used values.
7. The process is repeated over multiple iterations (known as *epochs*) until the loss is minimized and the model predicts acceptably accurately.

Note

While it's easier to think of each case in the training data being passed through the network one at a time, in reality the data is batched into matrices and processed using linear algebraic calculations. For this reason, neural network training is best performed on computers with graphical processing units (GPUs) that are optimized for vector and matrix manipulation.

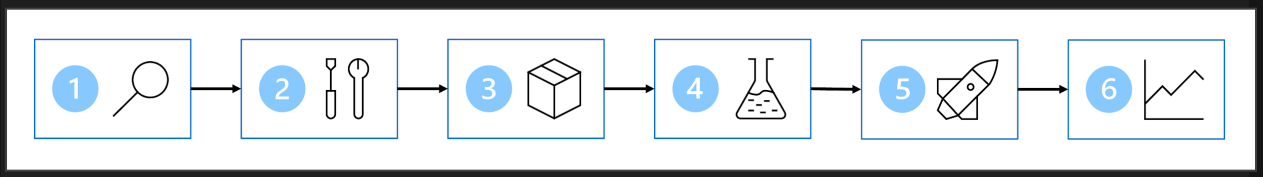
Get started with machine learning in Azure

Introduction

Thoughtfully designed machine learning solutions form the foundation of today's AI applications. From predictive analytics to personalized recommendations and beyond, machine learning solutions support the latest technological advances in society by using existing data to produce new insights.

Data scientists make decisions to tackle machine learning problems in different ways. The decisions they make affect the cost, speed, quality, and longevity of the solution.

In this module, you learn how to design an end-to-end machine learning solution with Microsoft Azure that can be used in an enterprise setting. Using the following six steps as a framework, we explore how to plan, train, deploy, and monitor machine learning solutions.



1. Define the problem: Decide on what the model should predict and when it's successful.
2. Get the data: Find data sources and get access.
3. Prepare the data: Explore the data. Clean and transform the data based on the model's requirements.
4. Train the model: Choose an algorithm and hyperparameter values based on trial and error.
5. Integrate the model: Deploy the model to an endpoint to generate predictions.

6. Monitor the model: Track the model's performance.

Note

The diagram is a simplified representation of the machine learning process. Typically, the process is iterative and continuous. For example, when monitoring the model you may decide to go back and retrain the model.

Next, let's look at how we can get started on a machine learning solution by defining the problem.

Define the problem

Completed

100 XP

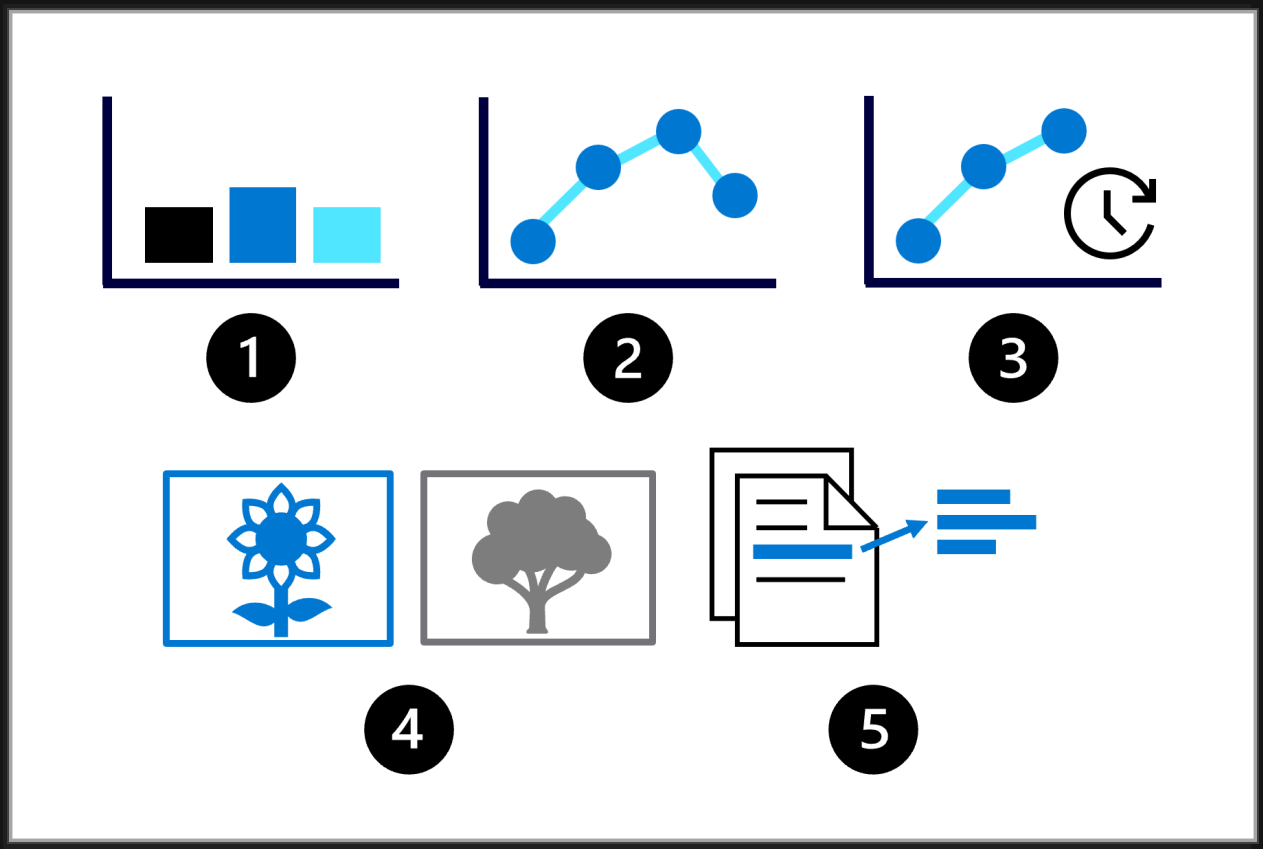
4 minutes

Starting with the first step, you want to define the problem the model should solve, by understanding:

- What the model's output should be.
- What type of machine learning task you use.
- What criteria make a model successful.

Depending on the data you have and the expected output of the model, you can identify the machine learning task. The task determines which types of algorithms you can use to train the model.

Some common machine learning tasks are:



1. Classification: Predict a categorical value.
2. Regression: Predict a numerical value.
3. Time-series forecasting: Predict future numerical values based on time-series data.
4. Computer vision: Classify images or detect objects in images.
5. Natural language processing (NLP): Extract insights from text.

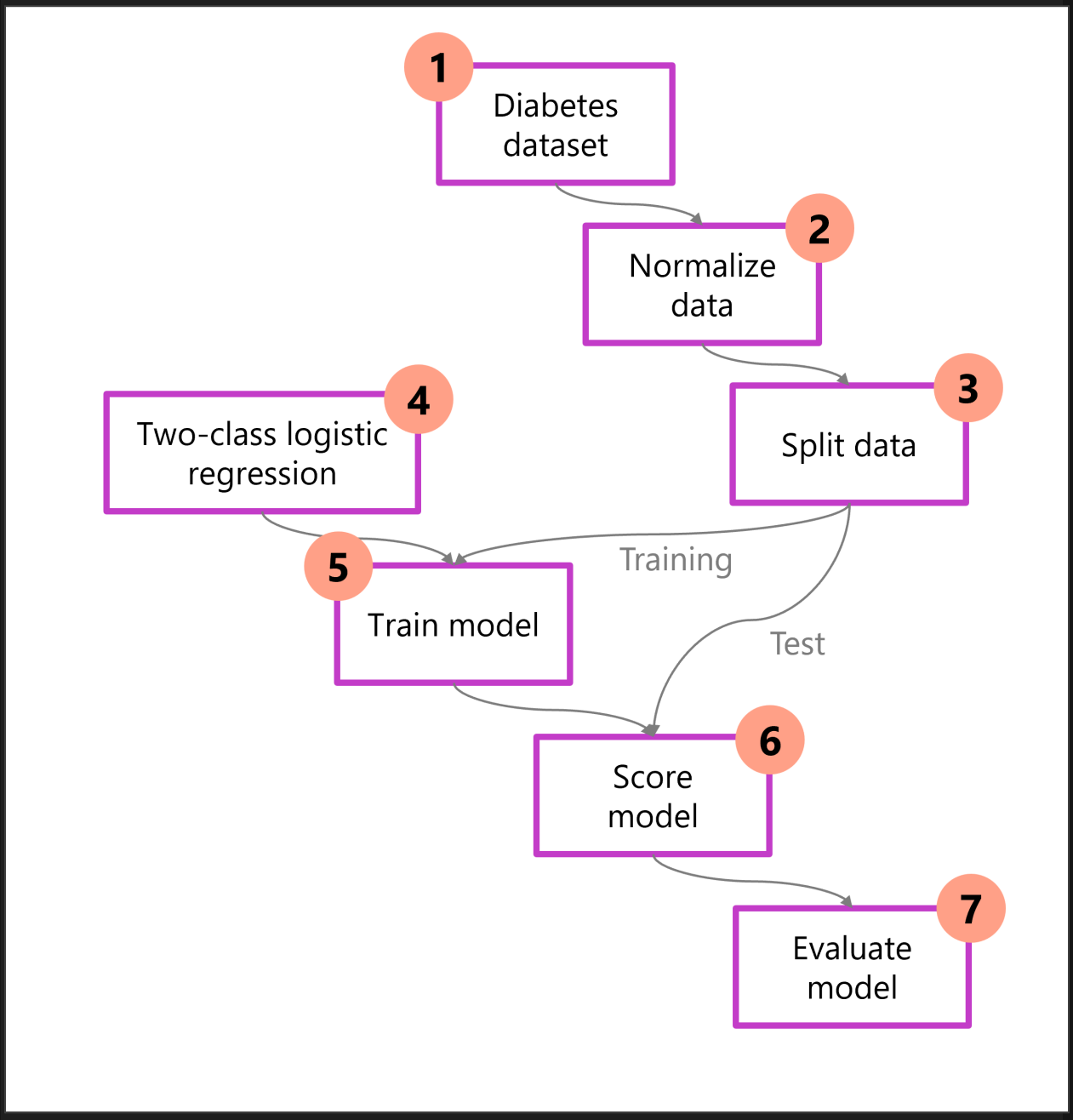
To train a model, you have a set of algorithms that you can use, depending on the task you want to perform. To evaluate the model, you can calculate performance metrics such as accuracy or precision. The metrics available also depend on the task your model needs to perform and help you to decide whether a model is successful in its task.

Explore an example

Consider a scenario where you want to determine if patients have diabetes. The problem you're trying to solve and the type of data available determines the machine learning task you choose. In this case, the available data are other health data points from patients. We can represent the output we want as *categorical* information that

either the patient has diabetes or doesn't have diabetes. Thus, the machine learning task is *classification*.

Understanding the entire process before you start gives you an opportunity to map out the decisions you need to make to design a successful machine learning solution. Following, is a diagram showing one way to approach the problem of identifying diabetes in a patient. In the diagram, the data is prepped, split, and trained using specific algorithms. Afterward, the model is evaluated for quality.



1. Load data: Import and inspect the dataset.
2. Preprocess data: Normalize and clean for consistency.
3. Split data: Separate into training and test sets.
4. Choose model: Select and configure an algorithm.
5. Train model: Learn patterns from the training data.
6. Score model: Generate predictions on test data.
7. Evaluate: Calculate performance metrics.

Training a machine learning model is often an iterative process, where you go through each of these steps multiple times to find the best performing model. Next, let's examine the data preparation process for developing a machine learning solution.

Get and prepare data

Data is the foundation of machine learning. Both data quantity and data quality affect the model's accuracy.

To train a machine learning model, you need to:

- Identify data source and format.
- Choose how to serve data.
- Design a data ingestion solution.

To get and prepare the data you use to train the machine learning model, you need to extract data from a source and make it available to the Azure service you want to use to train models or make predictions.

Identify data source and format

First, you need to identify your data source and its current data format.

Identify the	Examples

Data source	For example, the data can be stored in a Customer Relationship Management (CRM) system, in a transactional database like an SQL database, or be generated by an Internet of Things (IoT) device.
Data format	You need to understand the current format of the data, which can be tabular or structured data, semi-structured data or unstructured data.

Then, you need to decide what data you need to train your model, and in what format you want that data to be served to the model.

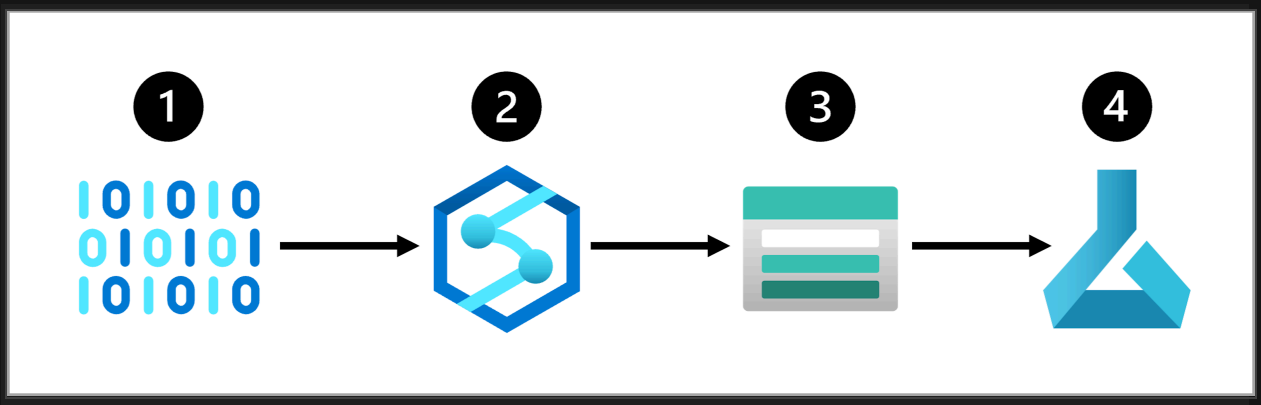
Design a data ingestion solution

In general, it's a best practice to extract data from its source before analyzing it. Whether you're using the data for data engineering, data analysis, or data science, you want to extract the data from its source, transform it, and load it into a serving layer. Such a process is also referred to as Extract, Transform, and Load (ETL) or Extract, Load, and Transform (ELT). The serving layer makes your data available for the service you use for further data processing like training machine learning models.

To move and transform data, you can use a data ingestion pipeline. A data ingestion pipeline is a sequence of tasks that move and transform the data. By creating a pipeline, you can choose to trigger the tasks manually or schedule the pipeline when you want the tasks to be automated. Such pipelines can be created with Azure services like Azure Synapse Analytics, Azure Databricks, and also Azure Machine Learning.

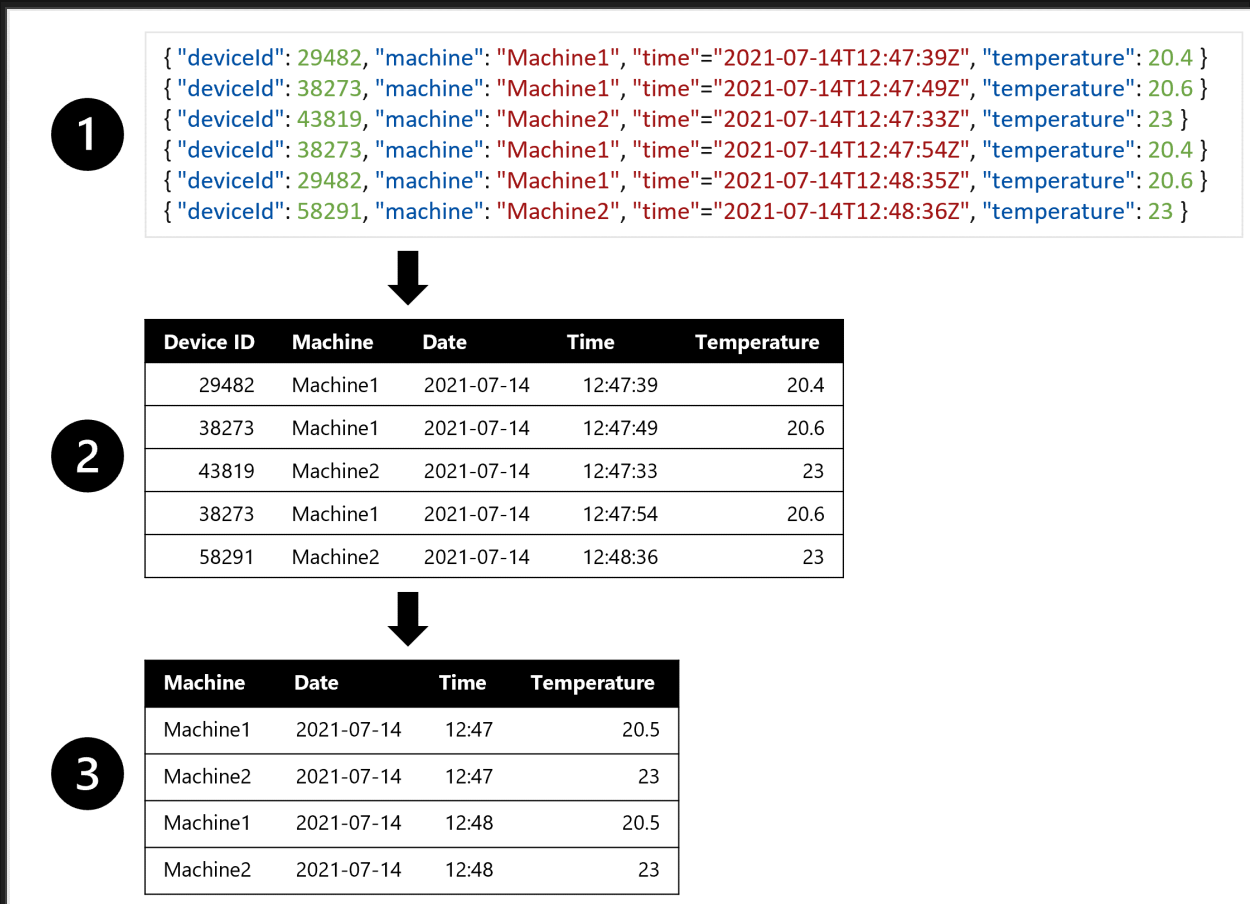
A common approach for a data ingestion solution is to:

1. Extract raw data from its source (like a CRM system or IoT device).
2. Copy and transform the data with Azure Synapse Analytics.
3. Store the prepared data in an Azure Blob Storage.
4. Train the model with Azure Machine Learning.



Explore an example

Imagine you want to train a weather forecasting model. You prefer one table in which all temperature measurements of each minute are combined. You want to create aggregates of the data and have a table of the average temperature per hour. To create the table, you want to transform the semi-structured data ingested from the IoT device that measures temperature at intervals, to tabular data.



For example, to create a dataset you can use to train the forecasting model, you can:

1. Extract data measurements as JSON objects from the IoT devices.
2. Convert the JSON objects to a table.
3. Transform the data to get the temperature per machine per minute.

Next, let's explore the services we can use to train machine learning models.

Train the model

There are many services available to train machine learning models. Which service you use depends on factors like:

- What type of model you need to train,
- Whether you need full control over model training,
- How much time you want to invest in model training,

- Which services are already within your organization,
- Which programming language you're comfortable with.

Within Azure, there are several services available for training machine learning models. Some commonly used services are:

Icon	Description
	Azure Machine Learning gives you many different options to train and manage your machine learning models. You can choose to work with the Studio for a UI-based experience, or manage your machine learning workloads with the Python SDK, or CLI for a code-first experience. Learn more about Azure Machine Learning.
	Azure Databricks is a data analytics platform that you can use for data engineering and data science. Azure Databricks uses distributed Spark compute to efficiently process your data. You can choose to train and manage models with Azure Databricks or by integrating Azure Databricks with other services such as Azure Machine Learning. Learn more about Azure Databricks.
	Microsoft Fabric is an integrated analytics platform designed to streamline data workflows between data analysts, data engineers, and data scientists. With Microsoft Fabric, you can prepare data, train a model, use the trained model to generate predictions, and visualize the data in Power BI reports. Learn more about Microsoft Fabric, and specifically about the data science features in Microsoft Fabric.

Azure AI Services is a collection of prebuilt machine learning models you can use for common machine learning tasks such as object detection in images. The models are offered as an application programming interface (API), so you can easily integrate a model with your application. Some models can be customized with your own training data, saving time and resources to train a new model from scratch. Learn more about Azure AI Services.

Features and capabilities of Azure Machine Learning

Let's focus on Azure Machine Learning. Microsoft Azure Machine Learning is a cloud service for training, deploying, and managing machine learning models. It's designed to be used by data scientists, software engineers, devops professionals, and others to manage the end-to-end lifecycle of machine learning projects.

Azure Machine Learning supports tasks including:

- Exploring data and preparing it for modeling.
- Training and evaluating machine learning models.
- Registering and managing trained models.
- Deploying trained models for use by applications and services.
- Reviewing and applying responsible AI principles and practices.

Azure Machine Learning provides the following features and capabilities to support machine learning workloads:

- Centralized storage and management of datasets for model training and evaluation.
- On-demand compute resources on which you can run machine learning jobs, such as training a model.
- Automated machine learning (AutoML), which makes it easy to run multiple training jobs with different algorithms and parameters to find the best model for your data.
- Visual tools to define orchestrated *pipelines* for processes such as model training or inferencing.

- Integration with common machine learning frameworks such as MLflow, which make it easier to manage model training, evaluation, and deployment at scale.
- Built-in support for visualizing and evaluating metrics for responsible AI, including model explainability, fairness assessment, and others.

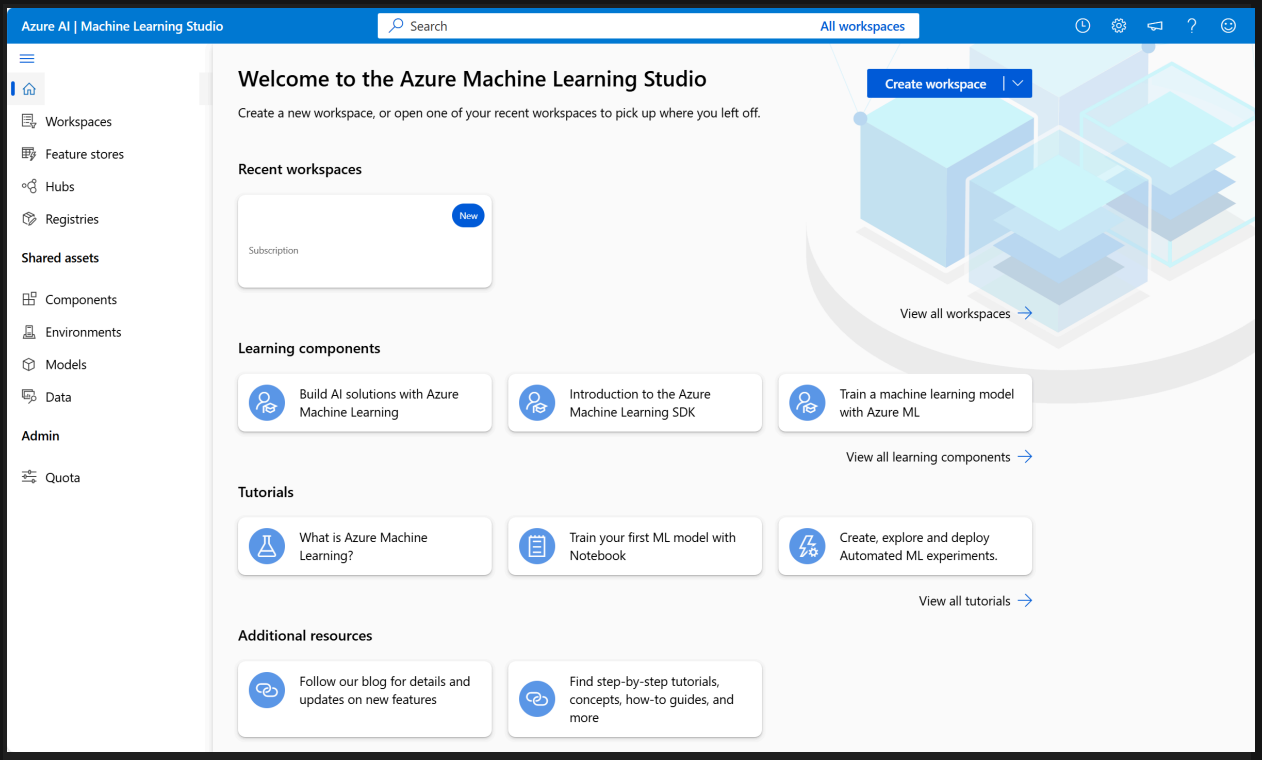
Next, let's see how we can get started with Azure Machine Learning in a user interface.

Use Azure Machine Learning studio

You can use Azure Machine Learning studio, a browser-based portal for managing your machine learning resources and jobs, to access many types of machine learning capabilities.

In Azure Machine Learning studio, you can (among other things):

- Import and explore data.
- Create and use compute resources.
- Run code in notebooks.
- Use visual tools to create jobs and pipelines.
- Use automated machine learning to train models.
- View details of trained models, including evaluation metrics, responsible AI information, and training parameters.
- Deploy trained models for on-request and batch inferencing.
- Import and manage models from a comprehensive model catalog.



Provisioning Azure Machine Learning resources

The primary resource required for Azure Machine Learning is an *Azure Machine Learning workspace*, which you can provision in an Azure subscription. Other supporting resources, including storage accounts, container registries, virtual machines, and others are created automatically as needed. You can create an Azure Machine Learning workspace in the *Azure portal*.

Decide between compute options

When you use Azure Machine Learning to train a model, you need to select compute. Compute refers to the computational resources required to perform the training process. Every time you train a model, you should monitor how long it takes to train the model and how much compute is used to execute your code. By monitoring the compute utilization, you know whether to scale your compute up or down.

When you choose to work with Azure instead of training a model on a local device, you have access to scalable and cost-effective compute.

Compute options	Considerations
Central Processing Unit (CPU) or a Graphics Processing Unit (GPU)	For smaller tabular datasets, a CPU is sufficient and cost-effective. For unstructured data like images or text, GPUs are more powerful and efficient. GPUs can also be used for larger tabular datasets, if CPU compute is proving to be insufficient.
General purpose or memory optimized	Use general purpose to have a balanced CPU-to-memory ratio, which is ideal for testing and development with smaller datasets. Use memory optimized to have a high memory-to-CPU ratio. Great for in-memory analytics, which is ideal when you have larger datasets or when you're working in notebooks.

Which compute options best fit your needs is often a case of trial and error. When running code, you should monitor the compute utilization to understand how much compute resources you're using. If training your model takes too long, even with the largest compute size, you can use GPUs instead of CPUs. Alternatively, you can choose to distribute model training by using Spark compute which require you to rewrite your training scripts.

Azure Automated Machine Learning

When you use Azure Machine Learning's Automated machine learning capabilities, you are automatically assigned compute. Azure Automated machine learning automates the time-consuming, iterative tasks of machine learning model development.

In Azure Machine Learning studio, you can use Automated machine learning to design and run your training experiments with the same steps described in this module, without needing to write code. Azure Automated machine learning provides a step-by-step wizard that helps you run machine learning training jobs. The automated training can be

used for many machine learning tasks, including regression, time-series forecasting, classification, computer vision, and natural language processing tasks. Within AutoML, you have access to your own datasets. Your trained machine learning models can be deployed as services.

Next, let's look at model deployment options.

Integrate a model

You should plan how you integrate the model, as it affects how you train the model or what training data you use. To integrate the model, you need to deploy a model to an endpoint. You can deploy a model to an endpoint for either real-time or batch predictions.

Deploy a model to an endpoint

When you train a model, the goal is often to integrate the model into an application.

To easily integrate a model into an application, you can use endpoints. Simply put, an endpoint can be a web address that an application can call to get a message back.

When you deploy a model to an endpoint, you have two options:

- Get real-time predictions
- Get batch predictions

Get real-time predictions

If you want the model to score any new data as it comes in, you need predictions in real-time.

Real-time predictions are often needed when a model is used by an application such as a mobile app or a website.

Imagine you have a website that contains your product catalog:

1. A customer selects a product on your website, such as a shirt.

2. Based on the customer's selection, the model recommends other items from the product catalog immediately. The website displays the model's recommendations.

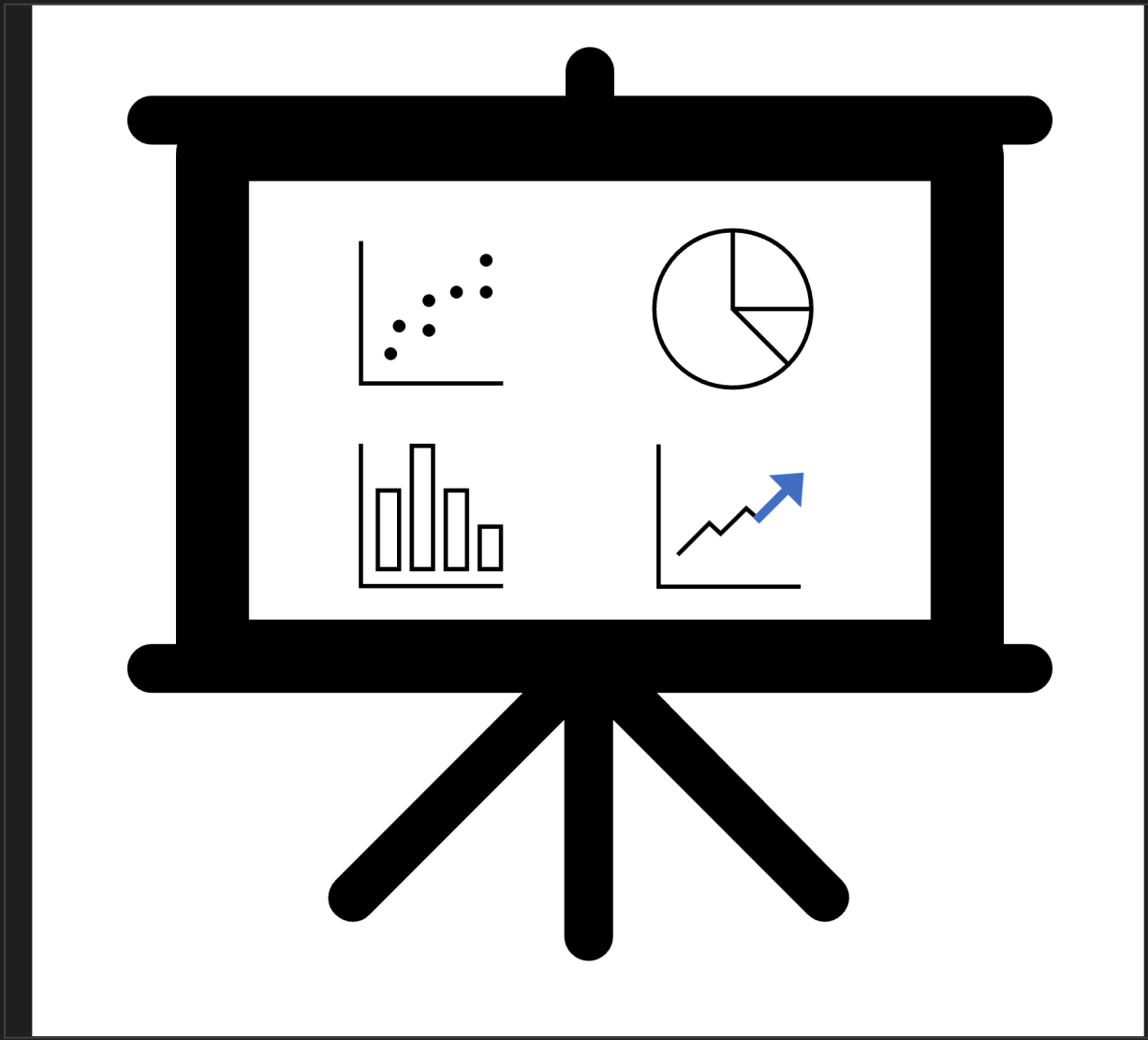
A customer can select a product in the web shop at any time. You want the model to find the recommendations almost immediately. The time it takes for the web page to load and display the shirt details is the time it should take to get the recommendations or predictions. Then, when the shirt is displayed, the recommendations can also be displayed.

Get batch predictions

If you want the model to score new data in batches, and save the results as a file or in a database, you need batch predictions.

For example, you can train a model that predicts orange juice sales for each future week. By predicting orange juice sales, you can ensure that supply is sufficient to meet expected demand.

Imagine you're visualizing all historical sales data in a report. You'll want to include the predicted sales in the same report.



Although orange juice is sold throughout the day, you only want to calculate the forecast once a week. You can collect the sales data throughout the week and call the model only when you have the sales data of a whole week. A collection of data points is referred to as a batch.

Decide between real-time or batch deployment

To decide whether to design a real-time or batch deployment solution, you need to consider the following questions:

- How often should predictions be generated?
- How soon are the results needed?
- Should predictions be generated individually or in batches?
- How much compute power is needed to execute the model?

Identify the necessary frequency of scoring

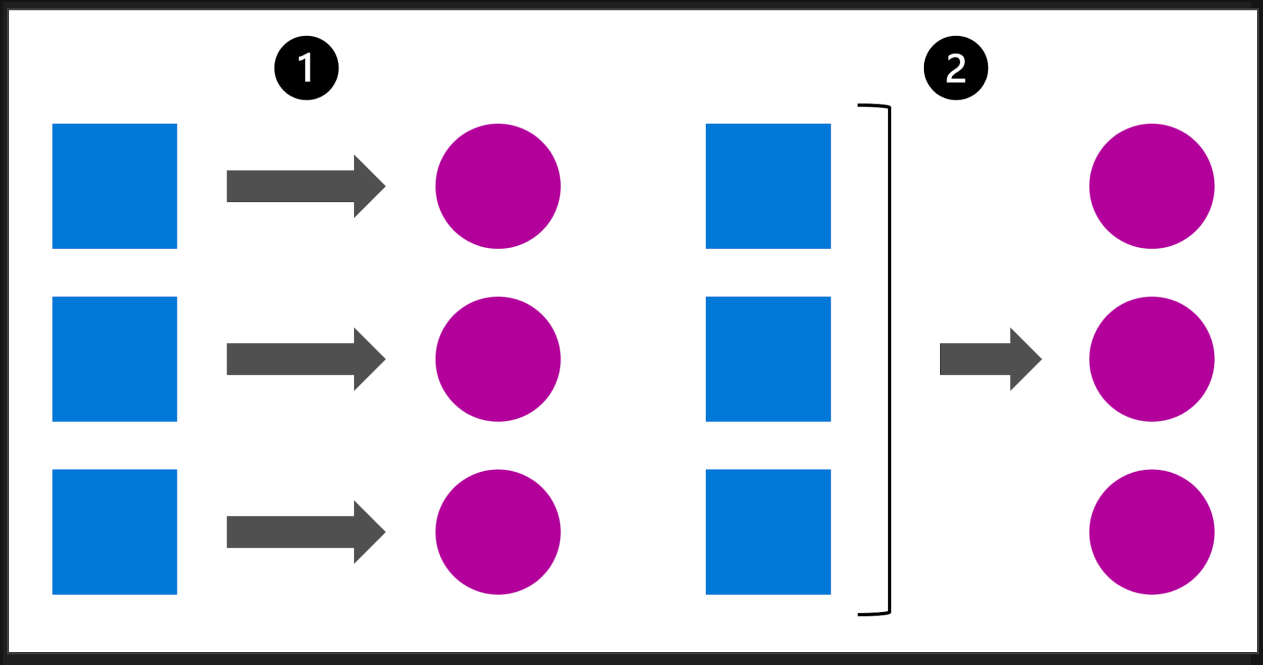
A common scenario is that you're using a model to score new data. Before you can get predictions in real-time or in batch, you must first collect the new data.

There are various ways to generate or collect data. New data can also be collected at different time intervals.

For example, you can collect temperature data from an Internet of Things (IoT) device every minute. You can get transactional data every time a customer buys a product from your web shop. Or you can extract financial data from a database every three months.

Generally, there are two types of use cases:

1. You need the model to score the new data as soon as it comes in.
2. You can schedule or trigger the model to score the new data that you've collected over time.



Whether you want real-time or batch predictions *doesn't necessarily depend on how often new data is collected*. Instead, it depends on how often and how quickly you need the predictions to be generated.

If you need the model's predictions immediately when new data is collected, you need real-time predictions. If the model's predictions are only consumed at certain times, you need batch predictions.

Decide on the number of predictions

Another important question to ask yourself is whether you need the predictions to be generated individually or in batches.

A simple way to illustrate the difference between individual and batch predictions is to imagine a table. Suppose you have a table of customer data where each row represents a customer. For each customer, you have some demographic data and behavioral data, such as how many products they've purchased from your web shop and when their last purchase was.

Based on this data, you can predict customer churn: whether a customer will buy from your web shop again or not.

Once you've trained the model, you can decide if you want to generate predictions:

- Individually: The model receives a *single row of data* and returns whether or not that individual customer will buy again.
- Batch: The model receives *multiple rows of data* in one table and returns whether or not each customer will buy again. The results are collated in a table that contains all predictions.

You can also generate individual or batch predictions when working with files. For example, when working with a computer vision model you may need to score a single image individually, or a collection of images in one batch.

Consider the cost of compute

In addition to using compute when training a model, you also need compute when deploying a model. Depending on whether you deploy the model to a real-time or batch endpoint, you'll use different types of compute. To decide whether to deploy your model to a real-time or batch endpoint, you must consider the cost of each type of compute.

If you need real-time predictions, you need compute that is always available and able to return the results (almost) immediately. Container technologies like *Azure Container Instance* (ACI) and *Azure Kubernetes Service* (AKS) are ideal for such scenarios as they provide a lightweight infrastructure for your deployed model.

However, when you deploy a model to a real-time endpoint and use such container technology, the compute is *always on*. Once a model is deployed, you're continuously paying for the compute as you can't pause, or stop the compute as the model must always be available for immediate predictions.

Alternatively, if you need batch predictions, you need compute that can handle a large workload. Ideally, you'd use a compute cluster that can score the data in *parallel* batches by using multiple nodes.

When working with compute clusters that can process data in parallel batches, the compute is provisioned by the workspace when the batch scoring is triggered, and scaled down to 0 nodes when there's no new data to process. By letting the workspace scale down an idle compute cluster, you can save significant costs.

Introduction to generative AI and agents

Introduction

Generative AI, and technologies that implement it are increasingly in the public consciousness – even among people who don't work in technology roles or have a background in computer science or machine learning. The futurist and novelist Arthur C. Clarke is quoted as observing that "any sufficiently advanced technology is indistinguishable from magic". In the case of generative AI, it does seem to have an almost miraculous ability to produce human-like original content, including poetry, prose, and even computer code.

However, there's no wizardry involved in generative AI – just the application of mathematical techniques incrementally discovered and refined over many years of research into statistics, data science, and machine learning. You can gain a high-level understanding of how the magic trick is done by learning the core concepts and principles explored in this module. As you learn more about the generative AI

technologies we have today, and how it powers a new generation of AI agents; you can help society imagine new possibilities for AI tomorrow.

Large language models (LLMs)

At the core of generative AI, large language models (LLMs) - and their more compact relations, small language models (SLMs) - encapsulate the linguistic and semantic relationships between the words and phrases in a vocabulary. The model can use these relationships to reason over natural language input and generate meaningful and relevant responses.

Fundamentally, LLMs are trained to generate *completions* based on *prompts*. Think of them as being super-powerful examples of the predictive text feature on many cellphones. A prompt starts a sequence of text predictions that results in a semantically correct completion. The trick is that the model understands the relationships between words and it can identify which words in the sequence so far are most likely to influence the next one; and use that to predict the most probable continuation of the sequence.

For example, consider the following sentence:

I heard a dog bark loudly at a cat

Now, suppose you only heard the first few words: *"I heard a dog ..."*. You know that some of these words are more helpful clues as to what the next word might be than others. You know that "heard" and "dog" are strong indicators of what comes next, and that helps you narrow down the probabilities. You know that there's a good chance the sentence will continue as *"I heard a dog bark"*.

You're able to guess the next word because:

- You have a large vocabulary of words to draw from.
- You've learned common linguistic structures, so you know how words relate to one another in meaningful sentences.
- You have an understanding of semantic concepts associated with words - you know that something you *heard* must be a sound of some kind, and you know that there are specific sounds that are made by a *dog*.

So how do we train a model to have these same abilities?

Tokenization

The first step is to provide the model with a large vocabulary of words and phrases; and we do mean *large*. The latest generation of LLMs have vocabularies that consist of hundreds of thousands of tokens, based on large volumes of training data from across the Internet and other sources.

Wait a minute. *Tokens*?

While we tend to think of language in terms of *words*, LLMs break down their vocabulary into *tokens*. Tokens include words, but also *sub*-words (like the "un" in "unbelievable" and "unlikely"), punctuation, and other commonly used sequences of characters. The first step in training a large language model therefore is to break down the training text into its distinct tokens, and assign a unique integer identifier to each one, like this:

- I (1)
- heard (2)
- a (3)
- dog (4)
- bark (5)
- loudly (6)
- at (7)
- a (3) *already assigned*
- cat (8)

and so on.

As you add more training data, more tokens will be added to the vocabulary and assigned identifiers; so you might end up with tokens for words like *puppy*, *skateboard*, *car*, and others.

Note

In this simple example, we've tokenized the example text based on *words*. In reality there would also be sub-words, punctuation, and other tokens.

Transforming tokens with a *transformer*

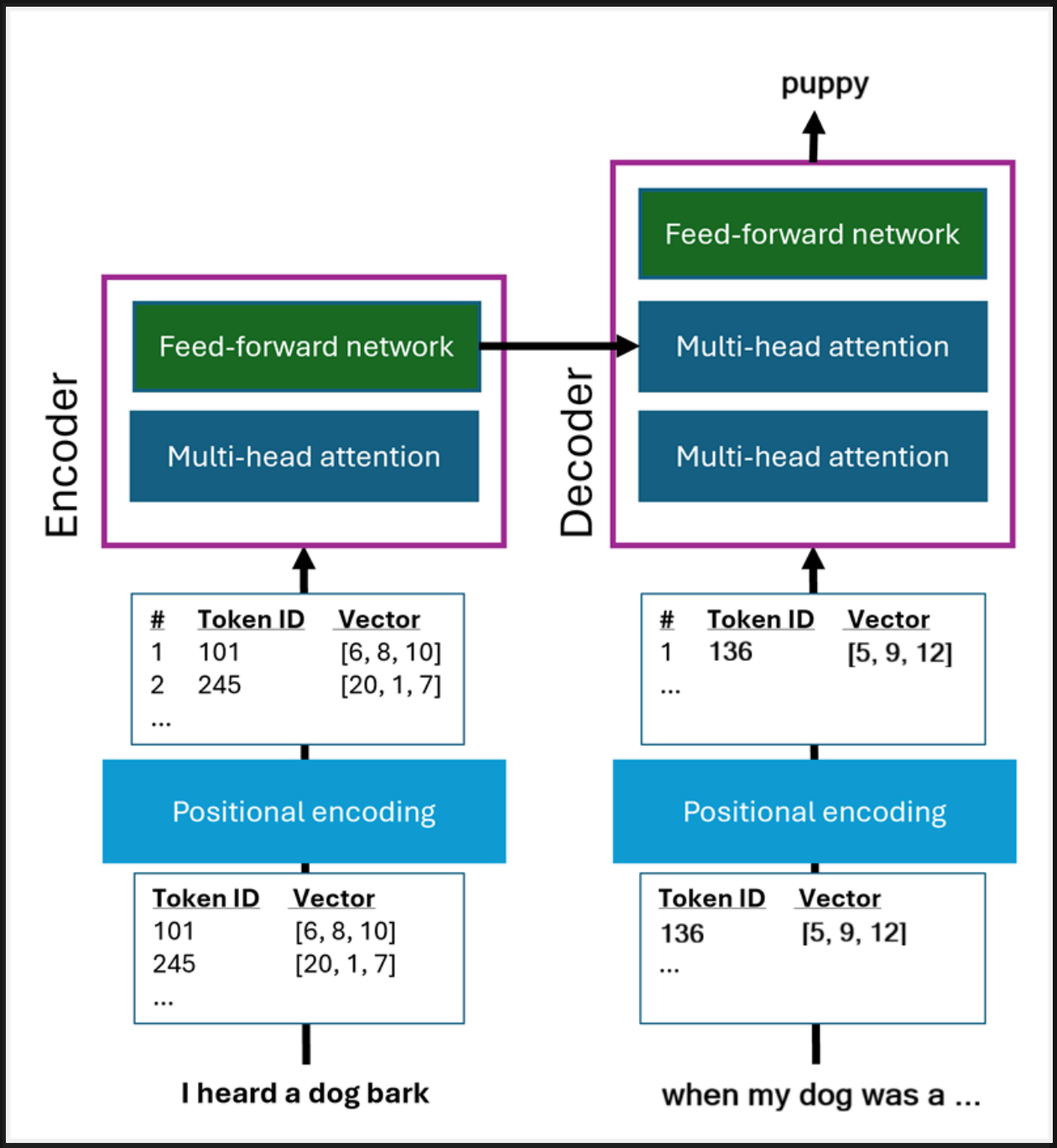
Now that we have a set of tokens with unique IDs, we need to find a way to relate them to one another. To do this, we assign each token a *vector* (an array of multiple numeric values, like [1, 23, 45]). Each vector has multiple numeric *elements* or *dimensions*, and we can use these to encode linguistic and semantic attributes of the token to help

provide a great deal of information about what the token *means* and how it relates to other tokens, in an efficient format.

We need to transform the initial vector representations of the tokens into new vectors with linguistic and semantic characteristics embedded in them, based on the contexts in which they appear in the training data. Because the new vectors have semantic values embedded in them, we call them *embeddings*.

To accomplish this task, we use a *transformer* model. This kind of model consists of two "blocks":

- An *encoder* block that creates the embeddings by applying a technique called *attention*. The attention layer examines each token in turn, and determines how it's influenced by the tokens around it. To make the encoding process more efficient, *multi-head* attention is used to evaluate multiple elements of the token in parallel and assign weights that can be used to calculate the new vector element values. The results of the attention layer are fed into a fully connected neural network to find the best vector representation of the embedding.
- A *decoder* layer that uses the embeddings calculated by the encoder to determine the next most probable token in a sequence started by a prompt. The decoder also uses attention and a feed-forward neural network to make its predictions.



Note

We've greatly simplified the transformer architecture and process in the description and diagram. Don't worry too much about the specific details of how attention works - the key point is that it helps capture linguistic and semantic characteristics of each token based on the contexts in which it's used. If you want a deeper dive into the transformer architecture and how it uses attention, you can read the original *Attention is all you need* paper.

Initial vectors and positional encoding

Initially, the token vector values are assigned randomly, before being fed through the transformer to create embedding vectors. The token vectors are fed into the transformer along with a *positional encoding* that indicates where the token appears in the sequence of training text (we need to do this because the order in which tokens appear in the sequence is relevant to how they relate to one another). For example, our tokens might start off looking like this:

Token	Token ID	Position	Vector
I	1	1	[3, 7, 10]
heard	2	2	[2, 15, 1]
a	3	3	[9, 11, 1]
dog	4	4	[2, 7, 11]
bark	5	5	[9, 12, 0]
loudly	6	6	[3, 8, 13]
at	7	7	[5, 7, 10]
a	3	8	[9, 11, 1]

cat	8	9	[8, -6, 9]
...	...	...	...
puppy	127	45	[7, 7, -2]
car	128	56	[5, -5, 1]
skateboard	129	67	[4, 7, 14]

Note

We've kept things simple by using vectors with only three elements (which will help us visualize them in three-dimensions later). In reality, the vectors have thousands of elements.

Attention and embeddings

To determine the vector representations of tokens that include embedded contextual information, the transformer uses *attention* layers. An attention layer considers each token in turn, within the context of the sequence of tokens in which it appears. The tokens around the current one are weighted to reflect their influence and the weights are used to calculate the element values for the current token's embedding vector. For example, when considering the token "bark" in the context of "I heard a dog bark", the tokens for "heard" and "dog" will be assigned more weight than "I" or "a", since they're stronger indicators for "bark".

Initially, the model doesn't "know" which tokens influence others; but as it's exposed to larger volumes of text, it can iteratively learn which tokens commonly appear together, and start to find patterns that help assign values to the vector elements that reflect the linguistic and semantic characteristics of the tokens, based on their proximity and frequency of use together. The process is made more efficient by using *multi-head* attention to consider different elements of the vectors in parallel.

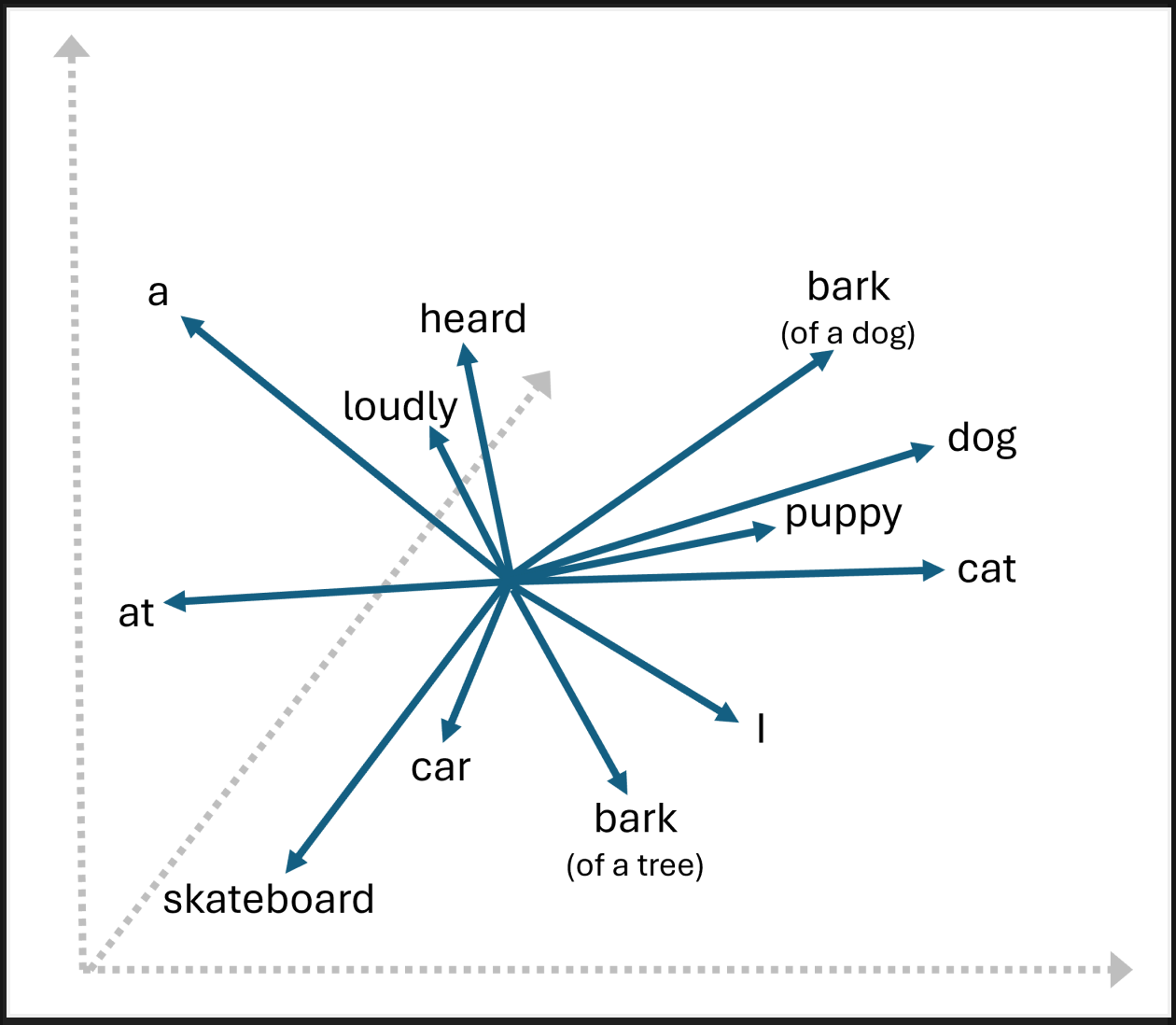
The result of the encoding process is a set of embeddings; vectors that include contextual information about how the tokens in the vocabulary relate to one another. A real transformer produces embeddings that include thousands of elements, but to keep things simple, let's stick to vectors with only three vectors in our example. The result of the encoding process for our vocabulary might look something like this:

Token	Token ID	Embedding
I	1	[2, 0, -1]
heard	2	[-2, 2, 4]
a	3	[-3, 5, 5]
dog	4	[10, 3, 2]
bark	5	[9, 2, 10]
loudly	6	[-3, 8, 3]
at	7	[-5, -1, 1]
cat	8	[10, 3, 1]
puppy	127	[5, 3, 2]

car	128	[-2, -2, 1]
skateboard	129	[-3, -2, 2]
bark	203	[2, -2, 3]

If you're observant, you might have spotted that our results include two embeddings for the token "bark". It's important to understand that the embeddings represent a token within a particular *context*; and some tokens might be used to mean multiple things. For example, the *bark* of a *dog* is different from the *bark* of a *tree*! Tokens that are commonly used in multiple contexts can produce multiple embeddings.

We can think of the elements of the embeddings as dimensions in a multi-dimensional vector-space. In our simple example, our embeddings only have three elements, so we can visualize them as vectors in three-dimensional space, like this:



Because the dimensions are calculated based on how the tokens relate linguistically to one another, tokens that are used in similar contexts (and therefore have similar meanings) result in vectors with similar directions. For example, the embeddings for "dog" and "puppy" point in more or less the same direction, which isn't too different from the embedding for "cat"; but very different from the embedding for "skateboard" or "car". We can measure how close tokens are to one another semantically by calculating the *cosine similarity* of their vectors.

Predicting completions from prompts

Now that we have a set of embeddings that encapsulate the contextual relationship between tokens, we can use the *decoder* block of a transformer to iteratively predict the next word in a sequence based on a starting *prompt*.

Once again, *attention* is used to consider each token in context; but this time the context to be considered can only include the tokens that *precede* the token we're trying to predict. The decoder model is trained, using data for which we already have the full sequence, by applying a technique called *masked attention*; in which the tokens after the current token are ignored. Since we already know the next token during training, the transformer can compare it to the predicted token and adjust the learned weights in later training iterations to reduce the error in the model.

When predicting a new completion, for which the next tokens are unknown, the attention layers calculate possible vectors for the next token and the feed-forward network is used to help determine the most probable candidate. The predicted value is then added to the sequence, and the whole process repeats to predict the *next* token; and so on, until the decoder predicts that the sequence has ended.

For example, given the sequence "*When my dog was a ...*", the model will evaluate the tokens in the sequence so far, use *attention* to assign weights, and predict that the next most probable token is "*puppy*" rather than, say, "*cat*" or "*skateboard*".

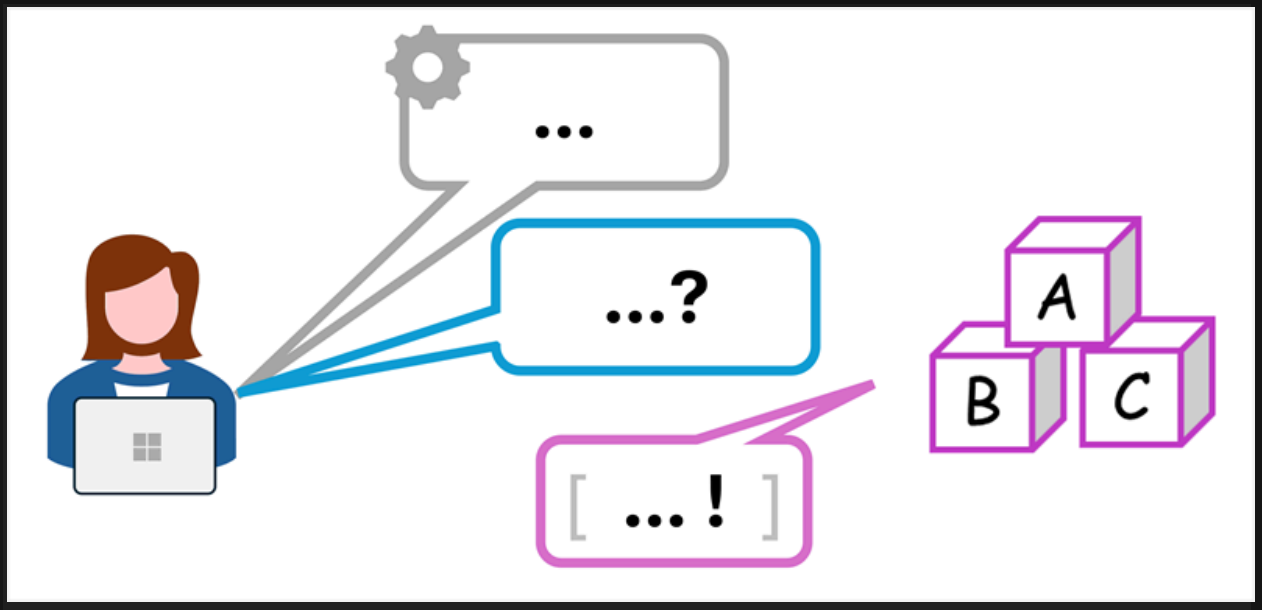
Prompts

A *prompt* is simply the input you give to an LLM to get a response. It might be a question or a command, or just a casual comment to start a conversation. The model responds to a prompt with a *completion*.

Types of prompt

There are two main types of prompts:

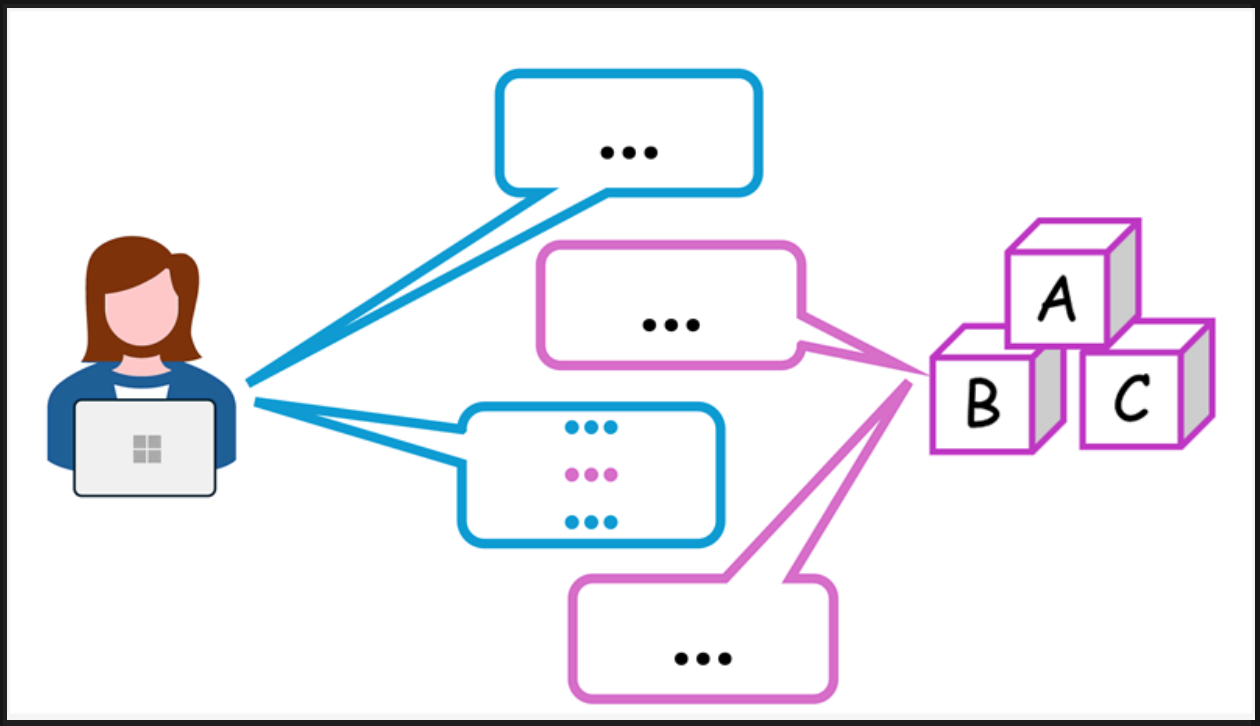
- System prompts that set the behavior and tone of the model, and any constraints it should adhere to. For example, "*You're a helpful assistant that responds in a cheerful, friendly manner.*". System prompts determine constraints and styles for the model's responses.
- User prompts that elicit a response to a specific question or instruction. For example, "*Summarize the key considerations for adopting generative AI described in GenAI_Considerations.docx for a corporate executive. Format the summary as no more than six bullet points with a professional tone.*".



Usually, the system prompt is set by the application that uses the model. User prompts can be entered by a human user in a chat application; or in some cases generated by the application on the user's behalf. The model responds to user prompts while obeying the overall guidance in the system prompt.

Conversation history

To keep a conversation consistent and relevant, generative AI apps often keep track of the conversation history; and include summarized versions of it in subsequent prompts. This ensures there's an ongoing context for the conversation that the model can build on.



For example, suppose the model responds to the system and user prompts described previously with the following completion:

Key considerations for adopting Generative AI include:

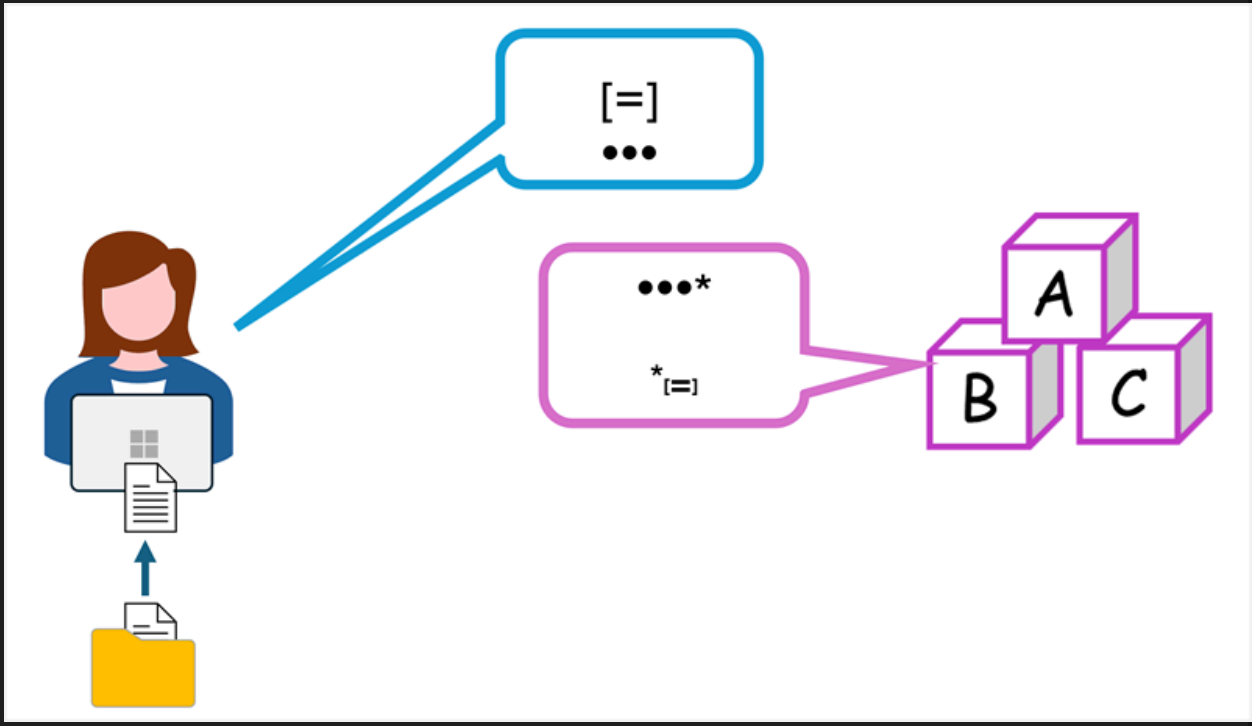
- *Establish Clear Governance and Responsible AI Policies*
- *Prioritize High-Value Use Cases Across Corporate Functions*
- *Mitigate Risks Around Privacy, Security, and Compliance*
- *Foster Organizational Readiness and Cultural Adaptation*
- *Measure Impact and Build Accountability*
- *Stay Ahead of Strategic and Competitive Shifts*

You could respond with a follow-up question, such as "*What are common privacy-related risks?*. The prompt would include the new question, but also the previous prompts and responses; providing context to the model so that it understands the question in relation to Generative AI adoption.

Retrieval augmented generation (RAG)

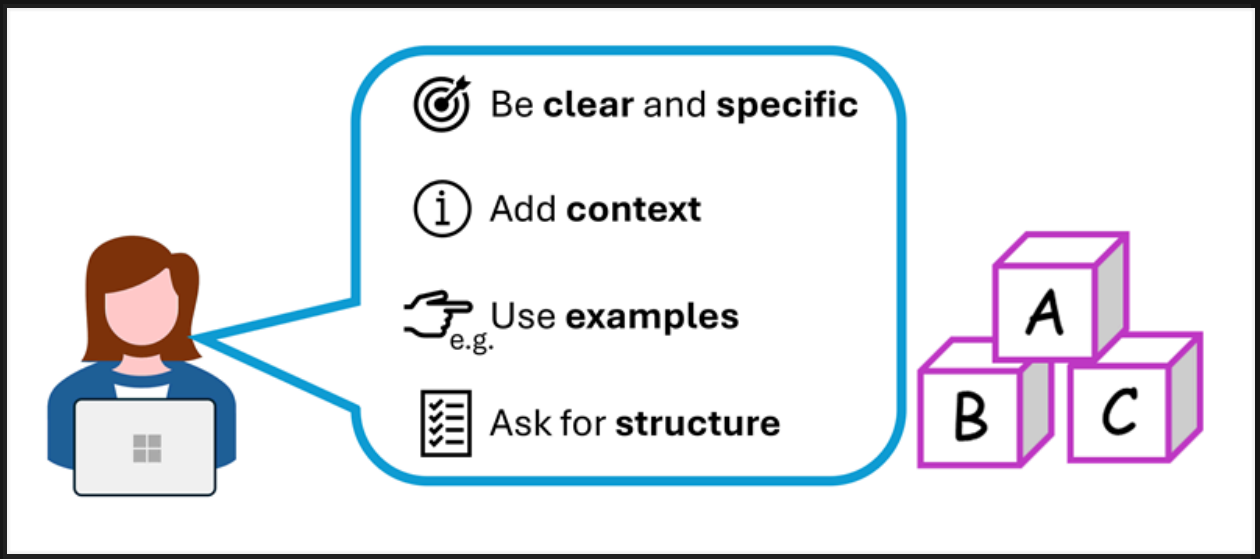
To add even more context, generative AI applications can use a technique called *retrieval augmented generation (RAG)*. This approach involves retrieving information, like documents or emails, and using it to augment the prompt with relevant data. The response generated by the model is then *grounded* in the information that was provided.

For example, suppose you submit a prompt like "*What's the maximum I can claim for travel expenses on a business trip?*". With no other information, a model will respond with a generic answer - probably telling you to consult your organization's expenses policy documentation. A better solution would be to build an expenses assistant app that initially queries the organization's expenses policy documentation, retrieving sections related to "travel expenses"; and then includes the retrieved information in the prompt that is sent to the model, along with your original question. Now the model can use the expenses policy information in the prompt to provide context, and respond with a more relevant answer.



Tips for better prompts

The quality of responses from generative AI assistants not only depends on the language model used, but on the prompts you submit to it.



To get better results from your prompts:

- Be clear and specific – prompts with explicit instructions or questions work better than vague language.
- Add context - mention the topic, audience, or format you want.
- Use examples, If you want a certain style, provide an example of what you mean.
- Ask for structure, Like bullet points, tables, or numbered lists.

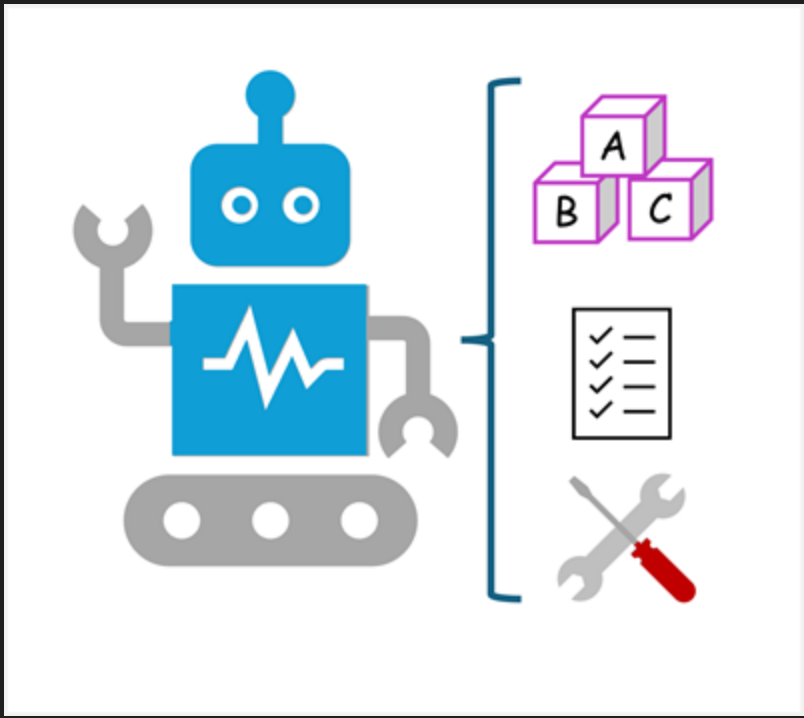
Using well-designed prompts can make a huge difference to the results you'll get from your generative AI model.

AI agents

Imagine having a digital assistant that doesn't just answer questions, but actually gets things done! Welcome to the world of AI agents.

Agents are software applications built on generative AI that can reason over and generate natural language, automate tasks by using tools, and respond to contextual conditions to take appropriate action.

Components of an AI agent



AI agents have three key elements:

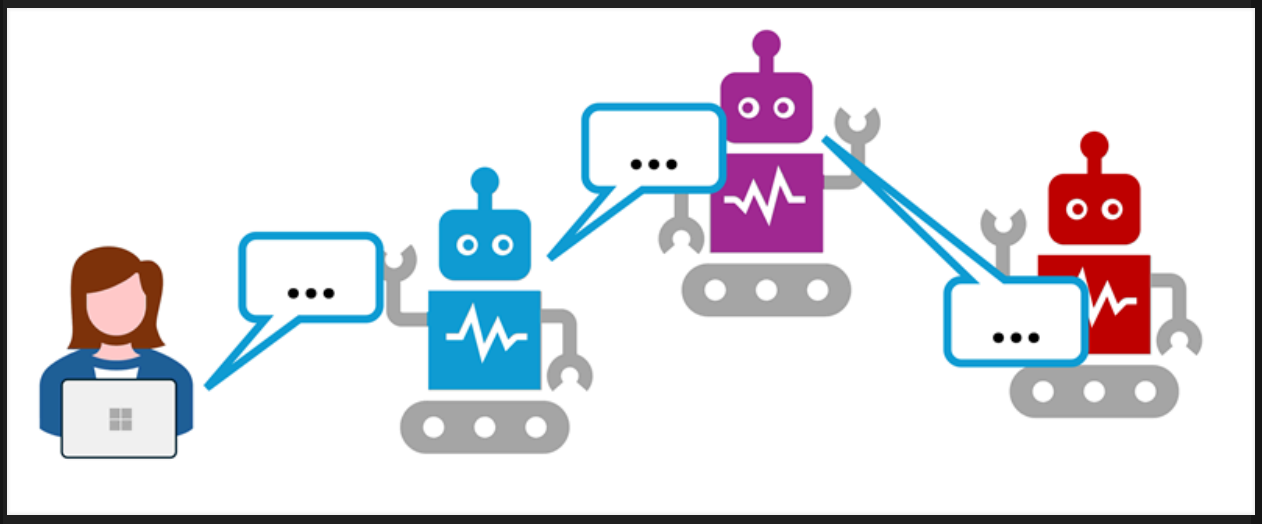
- A large language model: This is the agent's brain; using generative AI for language understanding and reasoning.
- Instructions: A system prompt that defines the agent's role and behavior. Think of it as the agent's job description.
- Tools: These are what the agent uses to interact with the world. Tools can include:
    - *Knowledge* tools that provide access to information, like search engines or databases.
    - *Action* tools that enable the agent to perform tasks, such as sending emails, updating calendars, or controlling devices.

With these capabilities, AI agents can take on the role of digital assistants that intelligently automate tasks and collaborate with you to work smarter and more efficiently.

Multi-agent systems

Agents can also work with one another, in multi-agent systems. Instead of one agent doing everything, multiple agents can collaborate—each with its own specialty. One might gather data, another might analyze it, and a third might take action. Together, they

form an AI-powered workforce that can handle complex workflows, just like a human team.



Agents communicate with each other through prompts, using generative AI to determine what tasks are required and which agents are responsible for completing them.

Agentic AI is set to be the next advance in how we use technology to find information and get work done.

Get started with generative AI in Azure

Introduction

In a short few years, generative AI, a subset of artificial intelligence that focuses on creating new content, has changed the way we work and revolutionized what is possible with technology. At times, the fast-moving developments in generative AI can feel challenging to keep track of even for seasoned developers.

In this module, gain a framework for understanding generative AI applications and how Microsoft Azure AI supports innovation. What does today's innovation look like? Consider these use cases:

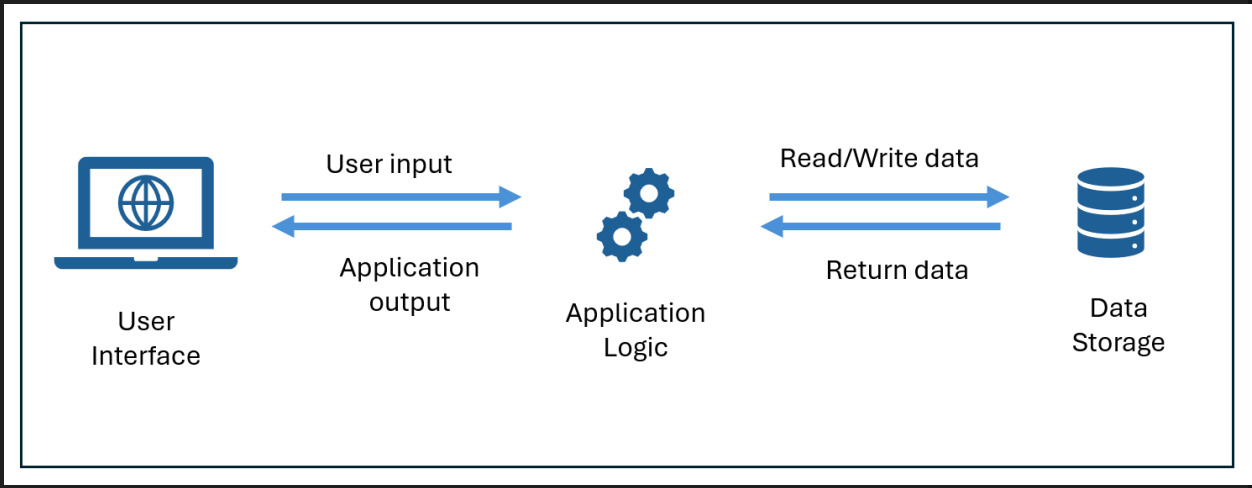
- *Marketing Content Creation*: Companies use Microsoft Copilot's generative AI to automatically write product descriptions, blog posts, and social media content—saving time and ensuring brand consistency across platforms.
- *Customer Support*: Businesses deploy AI-powered virtual agents that can understand and respond to customer inquiries in natural language, offering 24/7 support and reducing the load on human agents.

- *Code Generation*: Developers use tools like GitHub Copilot to generate code snippets, suggest functions, and even write entire modules based on natural language prompts, speeding up software development.
- *Image and Video Generation*: Designers and content creators use the latest models in Azure AI Foundry's model catalog to generate visuals for campaigns, storyboards, or concept art—often from just a text description.
- *Personalized Learning and Tutoring*: Educational platforms use generative AI to create custom quizzes, explanations, and study guides tailored to a student's learning style and progress.

Microsoft offers an ecosystem of tools for AI use and development. This module takes a closer look at tools for developers to build generative AI applications. Next, explore the landscape of generative AI applications.

Understand generative AI applications

Generative AI applications are built with language models. These language models power the 'app logic' component of the interaction between users and generative AI.



Understand assistants

Generative AI often appears as chat-based assistants that are integrated into applications to help users find information and perform tasks efficiently. One example of such an application is Microsoft Copilot, an AI-powered productivity tool designed to enhance your work experience by providing real-time intelligence and assistance.

Note

Microsoft Copilot is a generative AI based assistant that is integrated into a wide range of Microsoft applications and user experiences. Business users can use Microsoft

Copilot to boost their productivity and creativity with AI-generated content and automation of tasks. Developers can extend Microsoft Copilot by creating plug-ins that integrate Copilot into business processes and data, or even create copilot-like agents to build generative AI capabilities into apps and services. You can learn extensively about Microsoft Copilot here.

Understand agents

Generative AI that can execute tasks such as filing taxes or coordinating shipping arrangements, just as a few examples, are known as *agents*. Agents are applications that can respond to user input or assess situations *autonomously*, and take appropriate actions. These actions could help with a series of tasks. For example, an "executive assistant" agent could provide details about the location of a meeting on your calendar, then attach a map or automate the booking of a taxi or rideshare service to help you get there.

Agents contain three main components:

- A language model that powers reasoning and language understanding
- Instructions that define the agent's goals, behavior, and constraints
- Tools, or functions, that enable the agent to complete tasks

Note

Today's AI solutions often contain a combination of assistant, agentic, and other AI capabilities. The process of coordinating and managing multiple AI components—such as models, data sources, tools, and workflows—to work together efficiently in a unified solution is known as *orchestration*.

Use a framework for understanding generative AI applications

One way to think of different generative AI applications is by grouping them in buckets. In general, you can categorize industry and personal generative AI into three buckets, each requiring more customization: ready-to-use applications, extendable applications, and applications you build from the foundation.

Category	Description

Ready-to-use	These applications are ready-to-use generative AI applications. They do not require any programming work on the user's end to utilize the tool. You can start simply by asking the assistant a question.
Extendable	Some ready-to-use applications can also be extended using your own data. These customizations enable the assistant to better support specific business processes or tasks. Microsoft Copilot is an example of technology that is ready-to-use and extendable.
Applications you build from the foundation	You can build your own assistants and assistants with agentic capabilities starting from a language model.

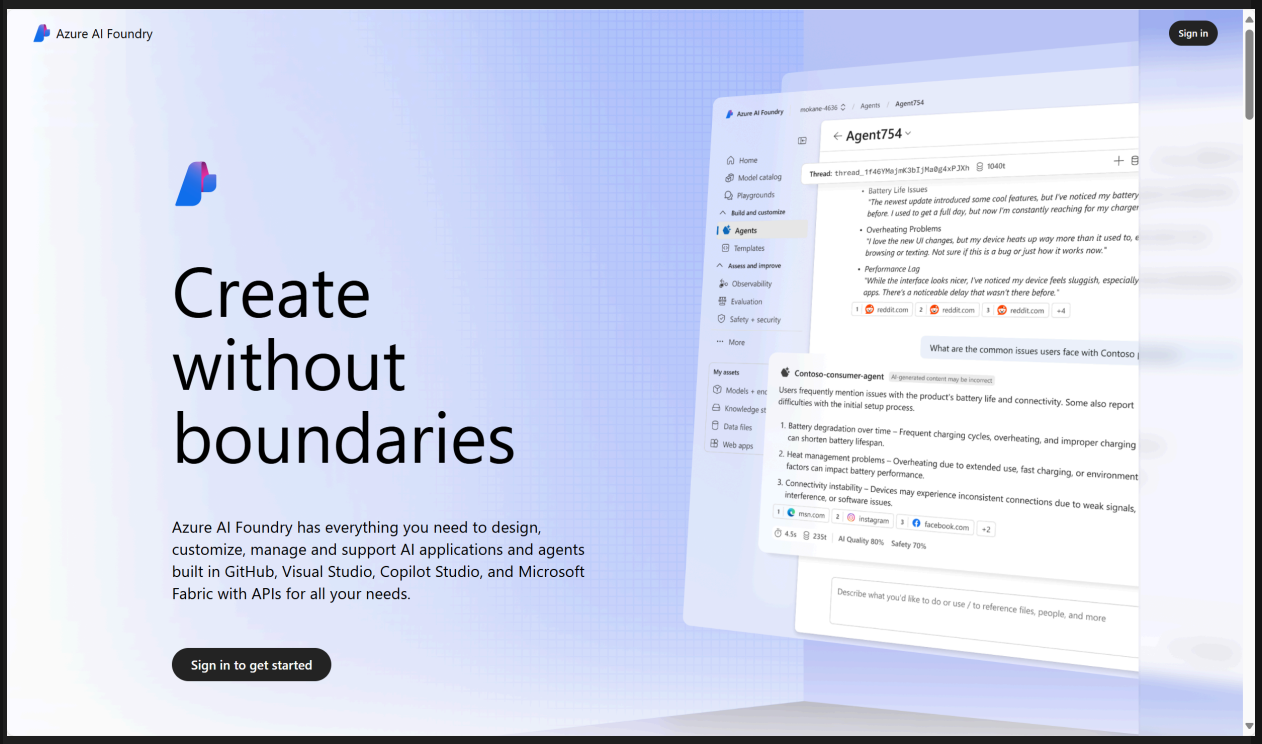
Often, you will use services to extend or build generative AI applications. These services provide the infrastructure, tools, and frameworks necessary to develop, train, and deploy generative AI models. For example, Microsoft provides services such as Copilot Studio to extend Microsoft 365 Copilot and Microsoft Azure AI Foundry to build AI from different models.

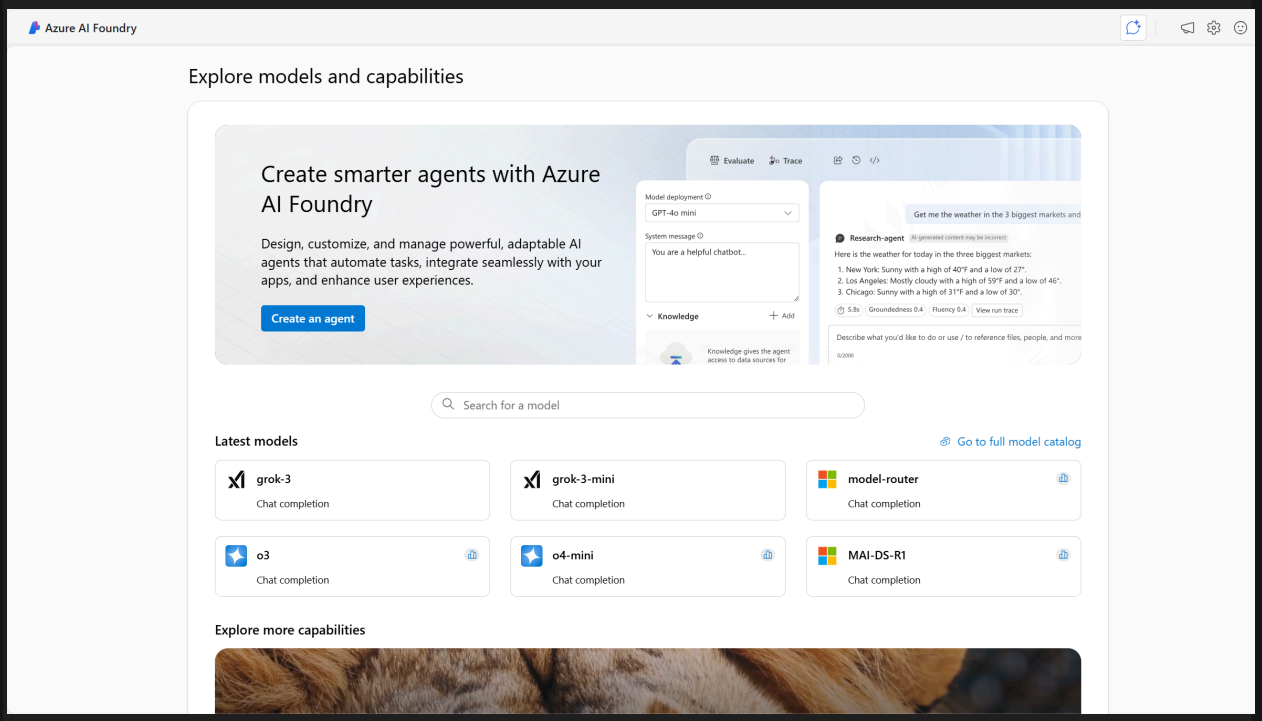
Next let's look at tools used to extend and build generative AI applications.

Understand tools to develop generative AI

Microsoft offers a powerful ecosystem of tools and services for building generative AI solutions, designed to support developers, data scientists, and enterprises at every stage of the AI lifecycle. You can develop generative AI solutions with several Microsoft solutions. This module will focus on Azure AI Foundry, Microsoft's unified platform for enterprise AI operations, model builders, and application development.

As a PaaS (platform as a service), Azure AI Foundry gives developers control over the customization of language models used for building applications. These models can be deployed in the cloud and consumed from custom-developed apps and services.





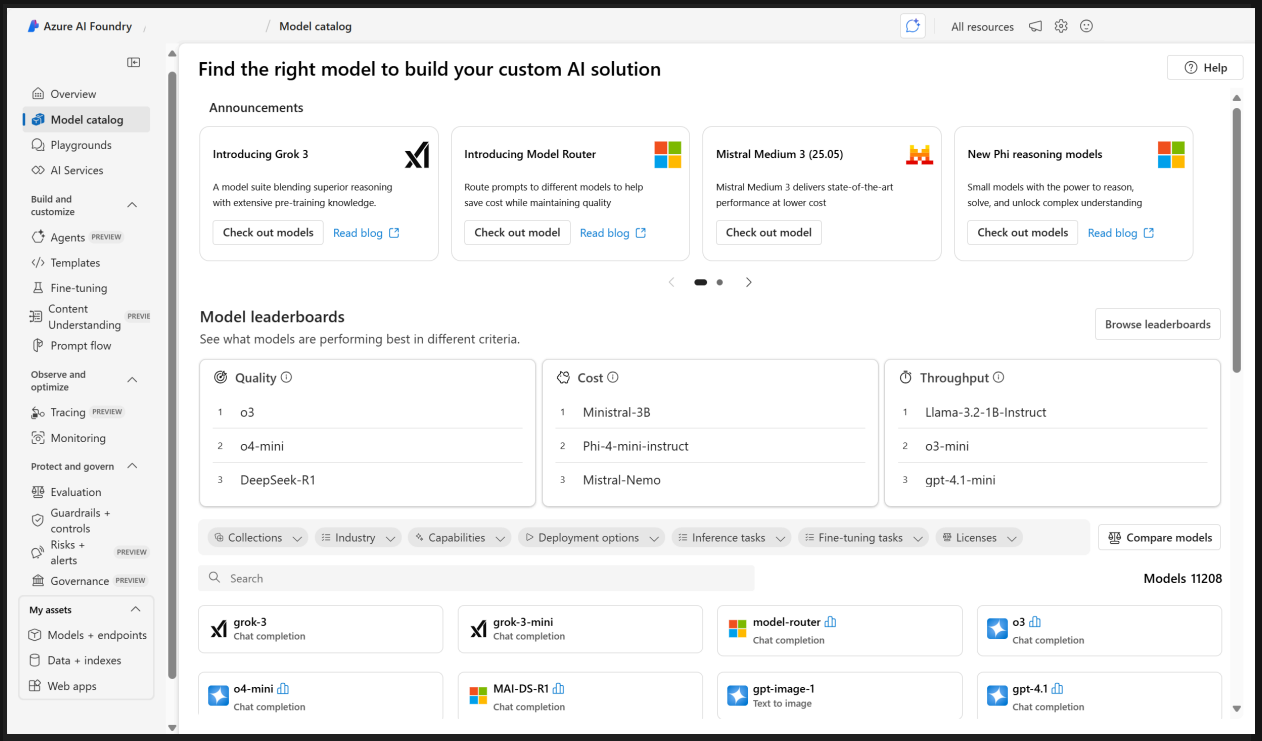
You can use Azure AI Foundry portal, a user interface for building, customizing, and managing AI applications and agents—especially those powered by generative AI.

Components of Azure AI Foundry include:

Component	Description
Azure AI Foundry model catalog	A centralized hub for discovering, comparing, and deploying a wide range of models for generative AI development.
Playgrounds	Ready-to-use environments for quickly testing ideas, trying out models, and exploring Azure AI services.
Azure AI services	In Azure AI Foundry portal, you can build, test, see demos, and deploy Azure AI services.
Solutions	You can build agents and customize models in Azure AI Foundry portal.
Observability	Ability to monitor usage and performance of your application's models.

Understand Azure AI Foundry's model catalog

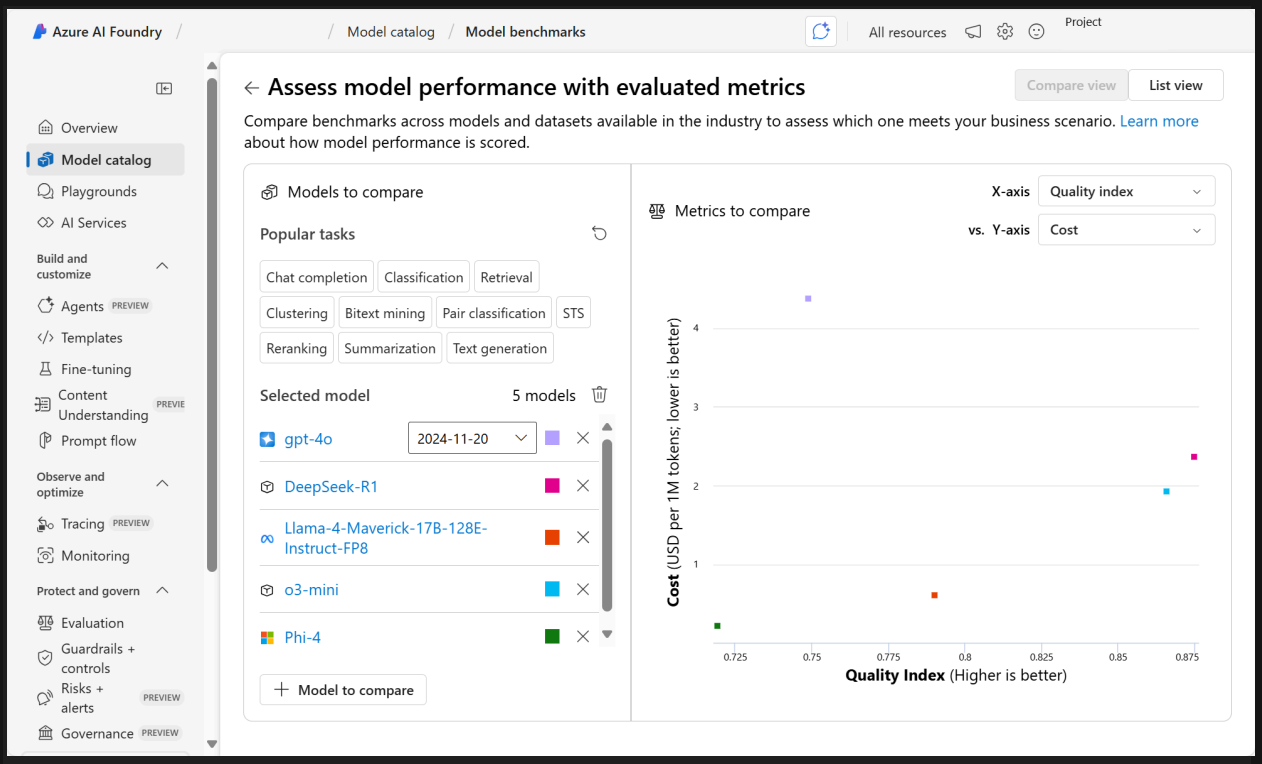
Azure AI Foundry provides a comprehensive and dynamic marketplace containing models sold directly by Microsoft and models from its partners and community.



Azure OpenAI in Foundry models make up Microsoft's first-party model family and are considered *foundation models*. Foundation models are pretrained on large texts and can be fine-tuned for specific tasks with a relatively small dataset.

You can deploy the models from Azure AI Foundry model catalog to an endpoint without any extra training. If you want the model to be specialized in a task, or perform better on domain-specific knowledge, you can also choose to customize a foundation model.

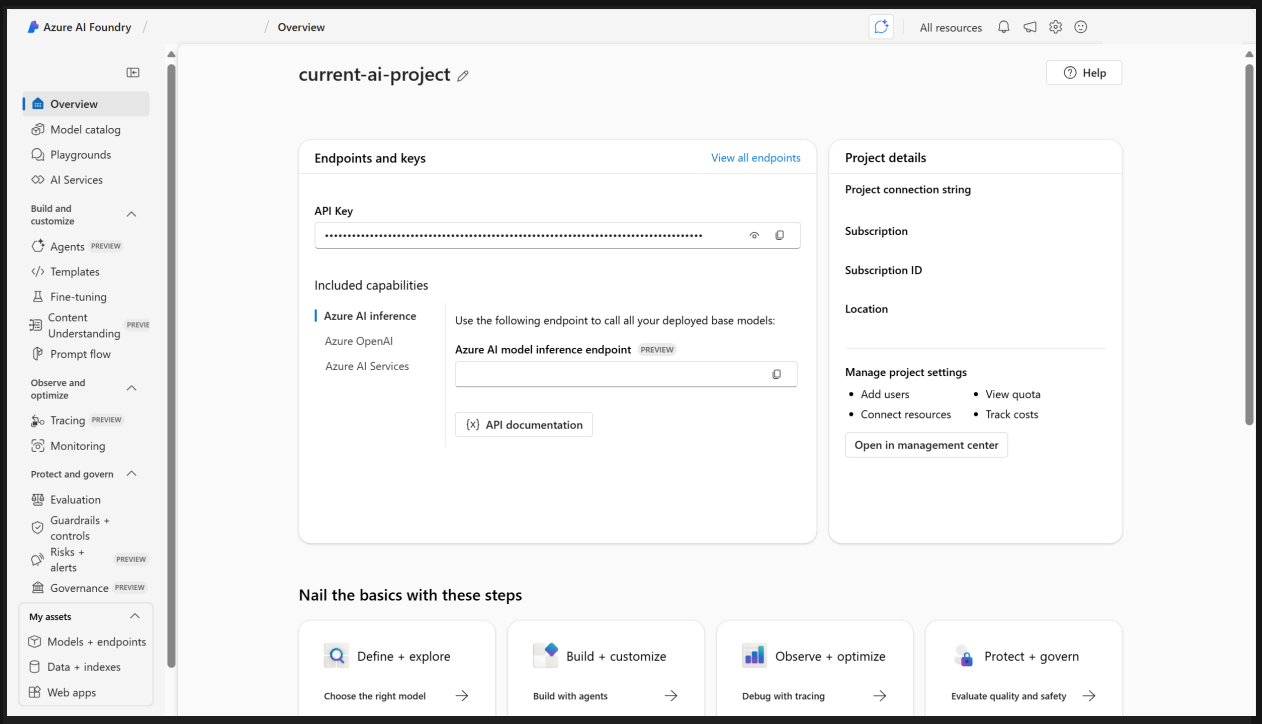
To choose the model that best fits your needs, you can test out different models in a *playground* setting and utilize *model leaderboards (preview)*. Model leaderboards provide a way to see what models are performing best in different criteria such as quality, cost, and throughput. You can also see graphical comparisons of models based on specific metrics.



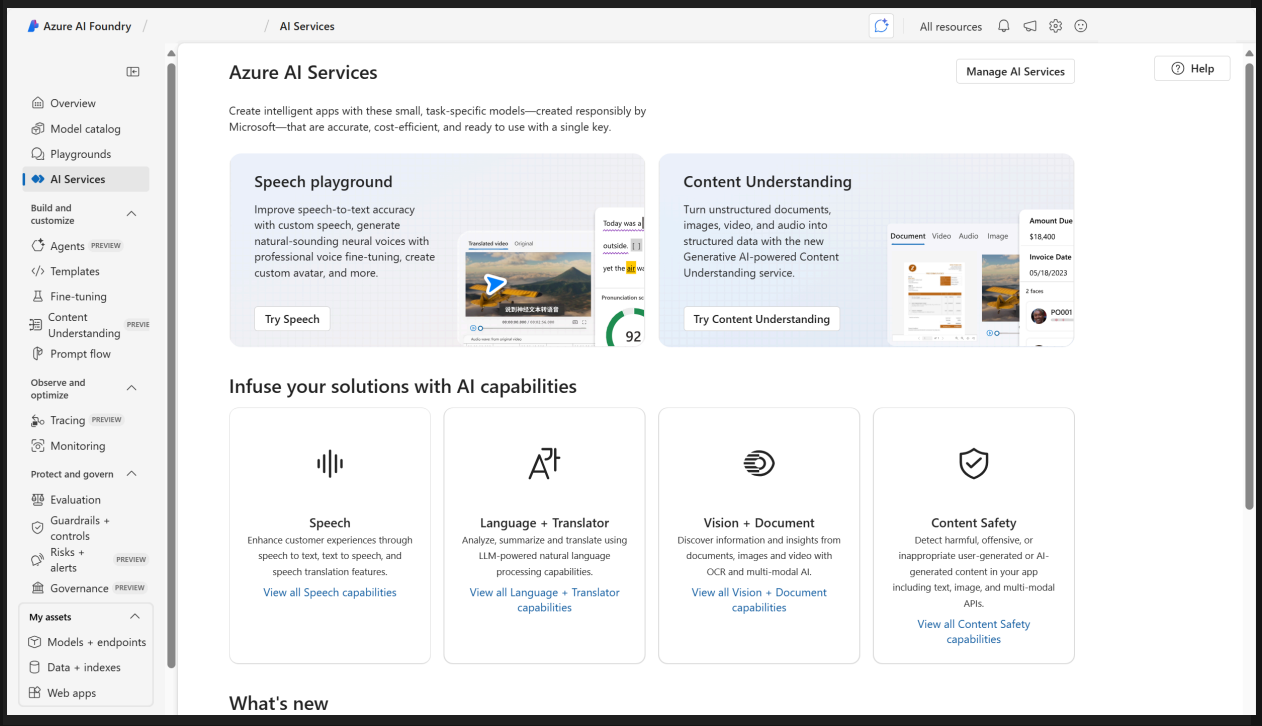
Next, let's take a closer look at how to get started with Azure AI Foundry capabilities.

Understand Azure AI Foundry capabilities

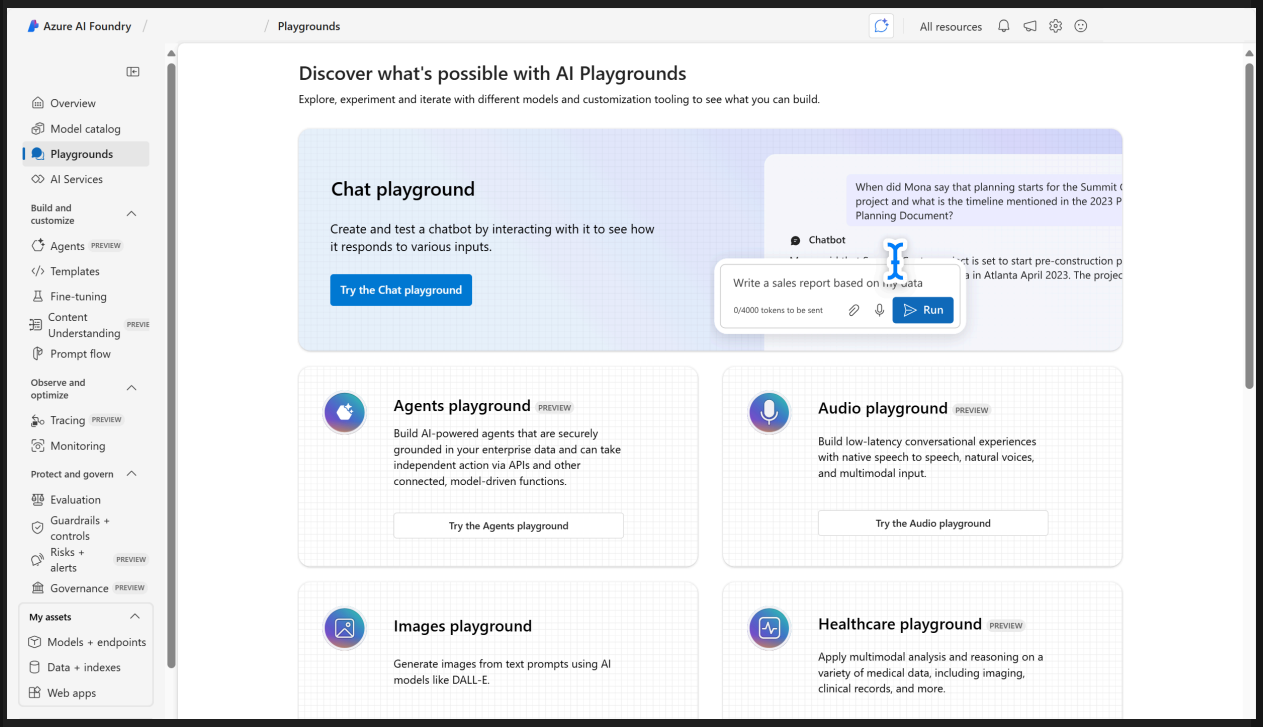
Azure AI Foundry portal provides a user interface based around hubs and projects. In general, creating a hub provides more comprehensive access to Azure AI and Azure Machine Learning. Within a hub, you can create projects. Projects provide more specific access to models and agent development. You can manage your projects from Azure AI Foundry portal's *overview* page.

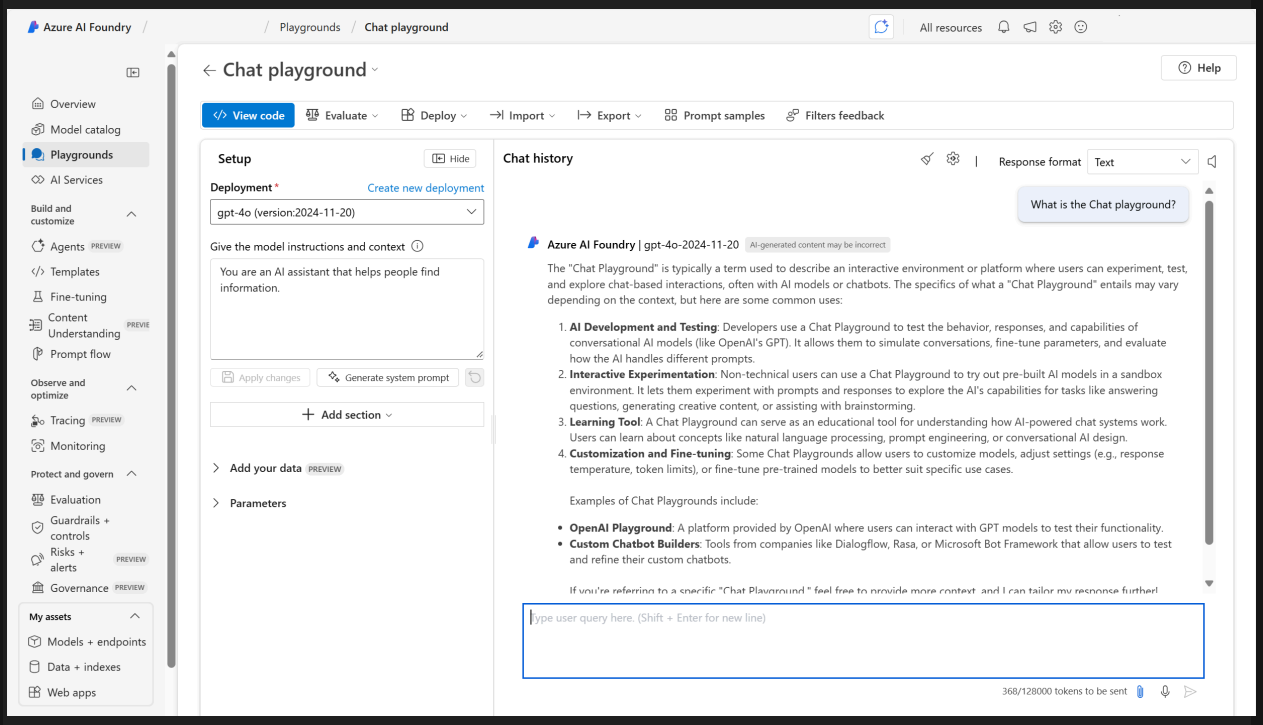


When you create an Azure AI Hub, several other resources are created in tandem, including an Azure AI services resource. In Azure AI Foundry portal, you can test all kinds of Azure AI services, including Azure AI Speech, Azure AI Language, Azure AI Vision, and Azure AI Foundry Content Safety.



In addition to demos, Azure AI Foundry portal provides playgrounds to test Azure AI services and other models from the model catalog.





Customizing models

There are many ways to customize the models in generative AI applications. The purpose of customizing your model is to improve aspects of its performance, including quality and safety of the responses. Let's take a look at four of the main ways you can customize models in Azure AI Foundry.

Method	Description
Using grounding data	Grounding refers to the process of ensuring that a system's outputs are aligned with factual, contextual, or reliable data sources. Grounding can be done in various ways, such as linking the model to a database, using search engines to retrieve real-time information, or incorporating domain-specific knowledge bases. The goal is to anchor the model's responses to these data sources, enhancing the trustworthiness and applicability of the generated content.
Implementing Retrieval-Augmented Generation (RAG)	RAG augments a language model by connecting it to an organization's proprietary database. This technique involves retrieving relevant information from a curated dataset and using it to generate contextually accurate responses. RAG enhances the model's performance by providing it with up-to-date and domain-specific information, which helps in generating more accurate and relevant answers. RAG is useful for applications where real-time access to dynamic data is crucial, such as customer support or knowledge management systems.

Fine-tuning	Involves taking a pretrained model and further training it on a smaller, task-specific dataset to make it more suitable for a particular application. This process allows the model to specialize and perform better at specific tasks that require domain-specific knowledge. Fine-tuning is useful for adapting models to domain-specific requirements, improving accuracy, and reducing the likelihood of generating irrelevant or inaccurate responses.
Managing security and governance controls	Security and governance controls are needed to manage access, authentication, and data usage. These controls help prevent the publication of incorrect or unauthorized information.

Next, let's understand how Azure AI Foundry provides tools for generative AI application performance evaluation.

Understand observability

There are many ways to measure generative AI's response quality. In general, you can think of three dimensions for evaluating and monitoring generative AI. These include:

- Performance and quality evaluators: assess the accuracy, groundedness, and relevance of generated content.
- Risk and safety evaluators: assess potential risks associated with AI-generated content to safeguard against content risks. This includes evaluating an AI system's predisposition towards generating harmful or inappropriate content.
- Custom evaluators: industry-specific metrics to meet specific needs and goals.

Azure AI Foundry supports observability features that improve the performance and trustworthiness of generative AI responses. *Evaluators* are specialized tools in Azure AI Foundry that measure the quality, safety, and reliability of AI responses.

Some evaluators include:

- *Groundedness*: measures how consistent the response is with respect to the retrieved context.
- *Relevance*: measures how relevant the response is with respect to the query.
- *Fluency*: measures natural language quality and readability.
- *Coherence*: measures logical consistency and flow of responses.
- *Content safety*: comprehensive assessment of various safety concerns.

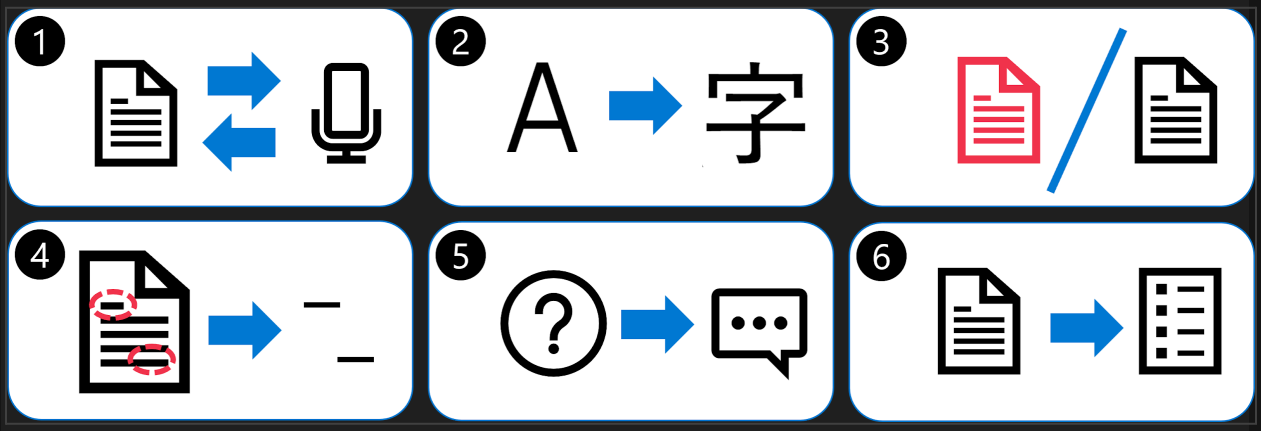
Next, let's try out generative AI capabilities in Azure AI Foundry portal.

Introduction to natural language processing concepts

Introduction

In order for computer systems to interpret the subject of a text in a similar way humans do, they use natural language processing (NLP), an area within AI that deals with understanding written or spoken language, and responding in kind. *Text analysis* describes NLP processes that extract information from unstructured text.

Some common NLP text analysis use cases are:



1. Speech-to-text and text-to-speech conversion. For example, generate subtitles for videos.
2. Machine translation. For example, translate text from English to Japanese.
3. Text classification. For example, label an email as spam or not spam.
4. Entity extraction. For example, extract keywords or names from a document.

5. Question answering. For example, provide answers to questions like "What is the capital of France?"
6. Text summarization. For example, generate a short one-paragraph summary from a multi-page document.

Historically, NLP has been challenging as our language is complex and computers find it hard to *understand* text. In this module, you learn how developments in AI and specifically NLP have led to the models we use today.

Next, let's examine some general principles and common techniques used to perform text analysis and other NLP tasks.

Understand how language is processed

Some of the earliest techniques used to analyze text with computers involve statistical analysis of a body of text (a *corpus*) to infer some kind of semantic meaning. Put simply, if you can determine the most commonly used words in a given document, you can often get a good idea of what the document is about.

Tokenization

The first step in analyzing a corpus is to break it down into *tokens*. For the sake of simplicity, you can think of each distinct word in the training text as a token, though in reality, tokens can be generated for partial words, or combinations of words and punctuation.

For example, consider this phrase from a famous US presidential speech: `"we choose to go to the moon"`. The phrase can be broken down into the following tokens, with numeric identifiers:

```
1. we

2. choose

3. to

4. go

5. the

6. moon
```

Notice that `"to"` (token number 3) is used twice in the corpus. The phrase `"we choose to go to the moon"` can be represented by the tokens {1,2,3,4,3,5,6}.

We've used a simple example in which tokens are identified for each distinct word in the text. However, consider the following concepts that may apply to tokenization depending on the specific kind of NLP problem you're trying to solve:

Concept	Description
Text normalization	Before generating tokens, you may choose to *normalize* the text by removing punctuation and changing all words to lower case. For analysis that relies purely on word frequency, this approach improves overall performance. However, some semantic meaning may be lost - for example, consider the sentence `"Mr Banks has worked in many banks."`. You may want your analysis to differentiate between the person `"Mr Banks"` and the `"banks"` in which he has worked. You may also want to consider `"banks."` as a separate token to `"banks"` because the inclusion of a period provides the information that the word comes at the end of a sentence
Stop word removal	Stop words are words that should be excluded from the analysis. For example, `"the"`, `"a"`, or `"it"` make text easier for people to read but add little semantic meaning. By excluding these words, a text analysis solution may be better able to identify the important words.
n-grams	Multi-term phrases such as `"I have"` or `"he walked"`. A single word phrase is a `unigram`, a two-word phrase is a `bi-gram`, a three-word phrase is a `tri-gram`, and so on. By considering words as groups, a machine learning model can make better sense of the text.

Stemming	A technique in which algorithms are applied to consolidate words before counting them, so that words with the same root, like `"power"`, `"powered"`, and `"powerful"`, are interpreted as being the same token.

Next, let's see how statistical techniques enable us to model language.

Understand statistical techniques for NLP

Two important statistical techniques that form the foundation of natural language processing (NLP) include: Naïve Bayes and Term Frequency - Inverse Document Frequency (TF-IDF).

Understanding Naïve Bayes

Naïve Bayes is a statistical technique that was first used for email filtering. To learn the difference between spam and not spam, two documents are compared. Naïve Bayes classifiers identify which tokens are correlated with emails labeled as spam. In other words, the technique finds which group of words only occurs in one type of document and not in the other. The group of words is often referred to as bag-of-words features.

For example, the words `miracle cure`, `lose weight fast`, and `anti-aging` may appear more frequently in spam emails about dubious health products than your regular emails.

Though Naïve Bayes proved to be more effective than simple rule-based models for text classification, it was still relatively rudimentary as only the presence (and not the position) of a word or token was considered.

Understanding TF-IDF

The Term Frequency - Inverse Document Frequency (TF-IDF) technique had a similar approach in that it compared the frequency of a word in one document with the frequency of the word in a whole corpus of documents. By understanding in which context a word was being used, documents could be classified based on certain topics. TF-IDF is often used for information retrieval, to help understand which relative words or tokens to search for.

Note

In the context of NLP, a corpus refers to a large and structured collection of text documents that is used for machine learning tasks. Corpora (plural of corpus) serve as essential resources for training, testing, and evaluating various NLP models.

For example, after tokenizing the words in `"we choose to go to the moon"`, you can perform some analysis to count the number of occurrences of each token. The most commonly used words (other than *stop words* such as `"a"`, `"the"`, and so on) can often provide a clue as to the main subject of a text corpus. For example, the most common words in the entire text of the `"go to the moon"` speech we considered previously include `"new"`, `"go"`, `"space"`, and `"moon"`. If we were to tokenize the text as `bi-grams` (word pairs), the most common `bi-gram` in the speech is `"the moon"`. From this information, we can easily surmise that the text is primarily concerned with space travel and going to the moon.

Simple frequency analysis in which you simply count the number of occurrences of each token can be an effective way to analyze a single document, but when you need to differentiate across multiple documents within the same corpus, you need a way to determine which tokens are most relevant in each document. *TF-IDF* calculates scores based on how often a word or term appears in one document compared to its more general frequency across the entire collection of documents. Using this technique, a high degree of relevance is assumed for words that appear frequently in a particular document, but relatively infrequently across a wide range of other documents.

Next, let's look at the deep learning techniques used to create today's semantic models.

Understand semantic language models

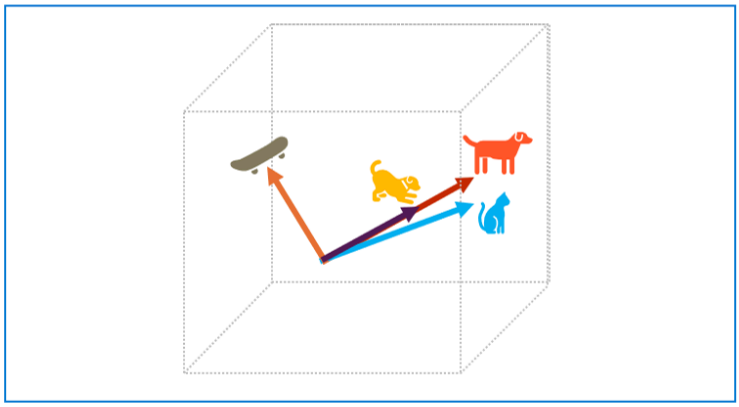
As the state of the art for NLP has advanced, the ability to train models that encapsulate the semantic relationship between tokens has led to the emergence of powerful deep learning language models. At the heart of these models is the encoding of language tokens as vectors (multi-valued arrays of numbers) known as *embeddings*.

Vectors represent lines in multidimensional space, describing direction and distance along multiple axes. Overall, the vector describes the direction and distance of the path from origin to end. Semantically similar tokens should result in vectors that have a similar orientation – in other words they point in the same direction. As a simple example, suppose the embeddings for our tokens consist of vectors with three elements, for example:

```
- 4 ("dog"): [10,3,2]

- 8 ("cat"): [10,3,1]

- 9 ("puppy") [5,2,1]
```

```
- 10 ("skateboard"): [-3,3,2]
```

In three-dimensional space, these vectors look like this:

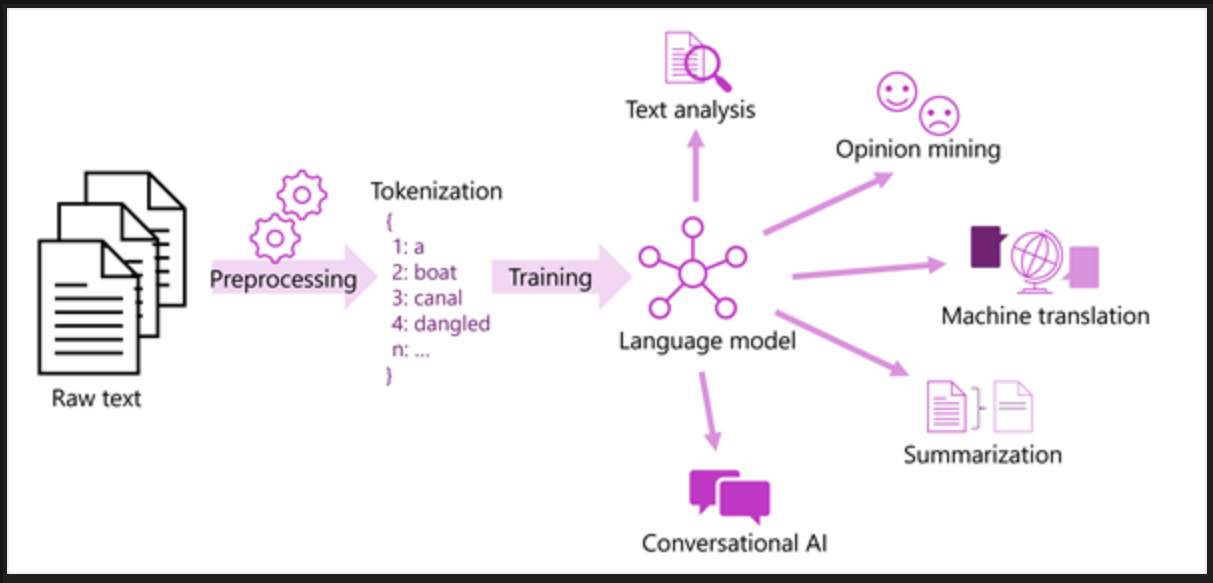


	Token	Word	Embedding
	4	dog	[10,3,2]
	8	cat	[10,3,1]
	9	puppy	[5,2,1]
	10	skateboard	[-3, 3, 2]

The embedding vectors for "dog" and "puppy" describe a path along an almost identical direction, which is also fairly similar to the direction for "cat". The embedding vector for "skateboard" however describes journey in a very different direction.

The language models we use in industry are based on these principles but have greater complexity. For example, the vectors used generally have many more dimensions. There are also multiple ways you can calculate appropriate embeddings for a given set of tokens. Different methods result in different predictions from natural language processing models.

A generalized view of most modern natural language processing solutions is shown in the following diagram. A large corpus of raw text is tokenized and used to train language models, which can support many different types of natural language processing task.



Machine learning for text classification

Another useful text analysis technique is to use a classification algorithm, such as *logistic regression*, to train a machine learning model that classifies text based on a known set of categorizations. A common application of this technique is to train a model that classifies text as *positive* or *negative* in order to perform *sentiment analysis* or *opinion mining*.

For example, consider the following restaurant reviews, which are already labeled as 0 (*negative*) or 1 (*positive*):

```
- *The food and service were both great*: 1

- *A really terrible experience*: 0

- *Mmm! tasty food and a fun vibe*: 1

- *Slow service and substandard food*: 0
```

With enough labeled reviews, you can train a classification model using the tokenized text as *features* and the sentiment (0 or 1) a *label*. The model will encapsulate a relationship between tokens and sentiment - for example, reviews with tokens for words like `"great"`, `"tasty"`, or `"fun"` are more likely to return a sentiment of 1 (*positive*), while reviews with words like `"terrible"`, `"slow"`, and `"substandard"` are more likely to return 0 (*negative*).

Get started with natural language processing in Azure

Introduction

Natural Language Processing (NLP) is a field of artificial intelligence focused on enabling machines to understand, interpret, and respond to human language. The goal of NLP is to analyze and extract meaning or structure from existing text.

Consider some of these applications of NLP:

- *Customer Feedback Analysis*: Organizations need to analyze large volumes of customer reviews, support tickets, or survey responses. By applying sentiment analysis and key phrase extraction, businesses can identify trends, detect dissatisfaction early, and improve customer experiences.
- *Healthcare Text Analysis*: In the healthcare sector, Azure's language solutions are used to extract clinical information from unstructured medical documents. Features like entity recognition and text analytics for health help identify symptoms, medications, and diagnoses, supporting faster and more accurate decision-making.
- *Conversational AI with Virtual Agents*: Azure's language solutions power virtual assistants that can interpret user intent, translate conversations, extract relevant entities, and respond appropriately.

Next, let's explore language capabilities on Azure.

Understand natural language processing on Azure

Core natural language processing (NLP) tasks include: language detection, sentiment analysis, named entity recognition, text classification, translation, and summarization. These tasks are supported by Azure AI services including:

Service	Description

 Azure AI Language service	A cloud-based service that includes features for understanding and analyzing text. Azure AI Language includes various features that support sentiment analysis, key phrase identification, text summarization, and conversational language understanding.
 Azure AI Translator service	A cloud-based service that uses Neural Machine Translation (NMT) for translation, which analyzes the semantic context of the text and renders a more accurate and complete translation as a result.

Next, let's explore Azure AI Language capabilities.

Understand Azure AI Language's text analysis capabilities

Azure AI Language is a part of the Azure AI services offerings that can perform advanced natural language processing over unstructured text. Azure AI Language's text analysis features include:

- Named entity recognition identifies people, places, events, and more. This feature can also be customized to extract custom categories.
- Entity linking identifies known entities together with a link to Wikipedia.
- Personal identifying information (PII) detection identifies personally sensitive information, including personal health information (PHI).
- Language detection identifies the language of the text and returns a language code such as "en" for English.
- Sentiment analysis and opinion mining identifies whether text is positive or negative.

- Summarization summarizes text by identifying the most important information.
- Key phrase extraction lists the main concepts from unstructured text.

Let's take a closer look at some of these features.

Entity recognition and linking

You can provide Azure AI Language with unstructured text and it returns a list of *entities* in the text that it recognizes. An entity is an item of a particular type or a category; and in some cases, subtype, for example:

Type	SubType	Example
Person		"Bill Gates", "John"
Location		"Paris", "New York"
Organization		"Microsoft"
Quantity	Number	"6" or "six"
Quantity	Percentage	"25%" or "fifty percent"
Quantity	Ordinal	"1st" or "first"
Quantity	Age	"90 day old" or "30 years old"

Quantity	Currency	"10.99"
Quantity	Dimension	"10 miles", "40 cm"
Quantity	Temperature	"45 degrees"
DateTime		"6:30PM February 4, 2012"
DateTime	Date	"May 2nd, 2017" or "05/02/2017"
DateTime	Time	"8am" or "8:00"
DateTime	DateRange	"May 2nd to May 5th"
DateTime	TimeRange	"6pm to 7pm"
DateTime	Duration	"1 minute and 45 seconds"
DateTime	Set	"every Tuesday"
URL		"`https://www.bing.com`"

Email	"support@microsoft.com"
US-based Phone Number	"(312) 555-0176"
IP Address	"10.0.1.125"

Azure AI Language also supports *entity linking* to help disambiguate entities by linking to a specific reference. For recognized entities, the service returns a URL for a relevant *Wikipedia* article.

For example, suppose you use Azure AI Language to detect entities in the following restaurant review extract:

"*I ate at the restaurant in Seattle last week.*"

Entity	Type	SubType	Wikipedia URL
Seattle	Location		https://en.wikipedia.org/wiki/Seattle
last week	DateTime	DateRange	

Language detection

You can identify the language in which text is written with Azure AI Language's language detection capability. For each document submitted the service detects:

- The language name (for example "English").
- The ISO 6391 language code (for example, "en").
- A score indicating a level of confidence in the language detection.

For example, consider a scenario where you own and operate a restaurant. Customers can complete surveys and provide feedback on the food, the service, staff, and so on. Suppose you received the following reviews from customers:

Review 1: "*A fantastic place for lunch. The soup was delicious.*"

Review 2: "*Comida maravillosa y gran servicio.*"

Review 3: "*The croque monsieur avec frites was terrific. Bon appetit!*"

You can use the text analytics capabilities in Azure AI Language to detect the language for each of these reviews; and it might respond with the following results:

Document	Language Name	ISO 6391 Code	Score
Review 1	English	en	1.0
Review 2	Spanish	es	1.0
Review 3	English	en	0.9

Notice that the language detected for review 3 is English, despite the text containing a mix of English and French. The language detection service focuses on the *predominant* language in the text. The service uses an algorithm to determine the predominant language, such as length of phrases or total amount of text for the language compared to other languages in the text. The predominant language is the value returned, along with the language code. The confidence score might be less than 1 as a result of the mixed language text.

There might be text that is ambiguous in nature, or that has mixed language content. These situations can present a challenge. An ambiguous content example would be a case where the document contains limited text, or only punctuation. For example, using Azure AI Language to analyze the text ":-)", results in a value of unknown for the language name and the language identifier, and a score of NaN (which is used to indicate *not a number*).

Sentiment analysis and opinion mining

The text analytics capabilities in Azure AI Language can evaluate text and return sentiment scores and labels for each sentence. This capability is useful for detecting positive and negative sentiment in social media, customer reviews, discussion forums and more.

Azure AI Language uses a prebuilt machine learning classification model to evaluate the text. The service returns sentiment scores in three categories: positive, neutral, and negative. In each of the categories, a score between 0 and 1 is provided. Scores indicate how likely the provided text is a particular sentiment. One document sentiment is also provided.

For example, the following two restaurant reviews could be analyzed for sentiment:

Review 1: "*We had dinner at this restaurant last night and the first thing I noticed was how courteous the staff was. We were greeted in a friendly manner and taken to our table right away. The table was clean, the chairs were comfortable, and the food was amazing.*"

and

Review 2: "*Our dining experience at this restaurant was one of the worst I've ever had. The service was slow, and the food was awful. I'll never eat at this establishment again.*"

The sentiment score for the first review might be: Document sentiment: positive Positive score: 0.90 Neutral score: 0.10 Negative score: 0.00

The second review might return a response: Document sentiment: negative Positive score: 0.00 Neutral score: 0.00 Negative score: 0.99

Key phrase extraction

Key phrase extraction identifies the main points from text. Consider the restaurant scenario discussed previously. If you have a large number of surveys, it can take a long time to read through the reviews. Instead, you can use the key phrase extraction capabilities of the Language service to summarize the main points.

You might receive a review such as:

"*We had dinner here for a birthday celebration and had a fantastic experience. We were greeted by a friendly hostess and taken to our table right away. The ambiance was relaxed, the food was amazing, and service was terrific. If you like great food and attentive service, you should try this place.*"

Key phrase extraction can provide some context to this review by extracting the following phrases:

- birthday celebration
- fantastic experience
- friendly hostess
- great food
- attentive service
- dinner
- table
- ambiance
- place

Next, let's look at Azure AI Language's conversational AI capabilities.

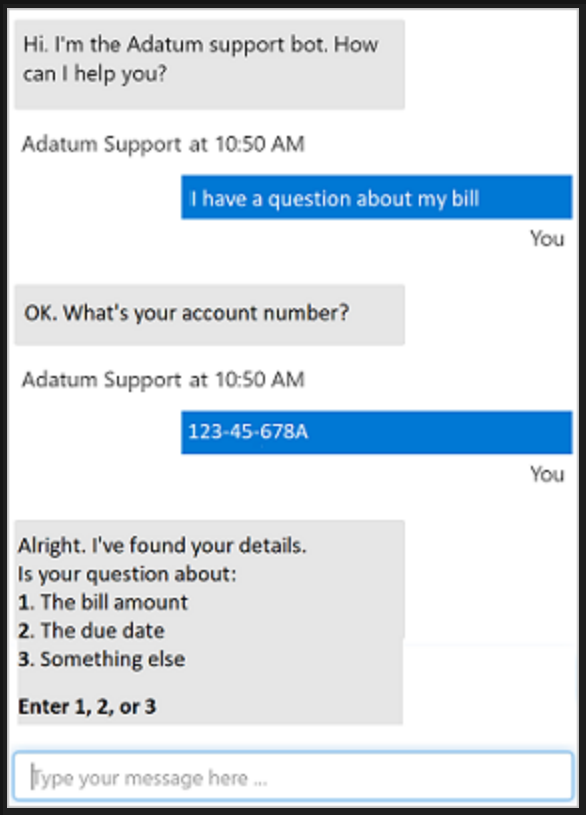
Azure AI Language's conversational AI capabilities

Azure AI Language also includes other features that encompass *conversational AI*. Conversational AI describes solutions that enable a dialog between AI and a human.

Question answering

Azure AI Language's question answering feature provides you with the ability to create conversational AI solutions. Question answering supports natural language AI workloads that require an automated conversational element. Typically, question answering is used to build bot applications that respond to customer queries. Question answering capabilities can respond immediately, answer concerns accurately, and interact with users in a natural multi-turned way. Bots can be implemented on a range of platforms, such as a web site or a social media platform.

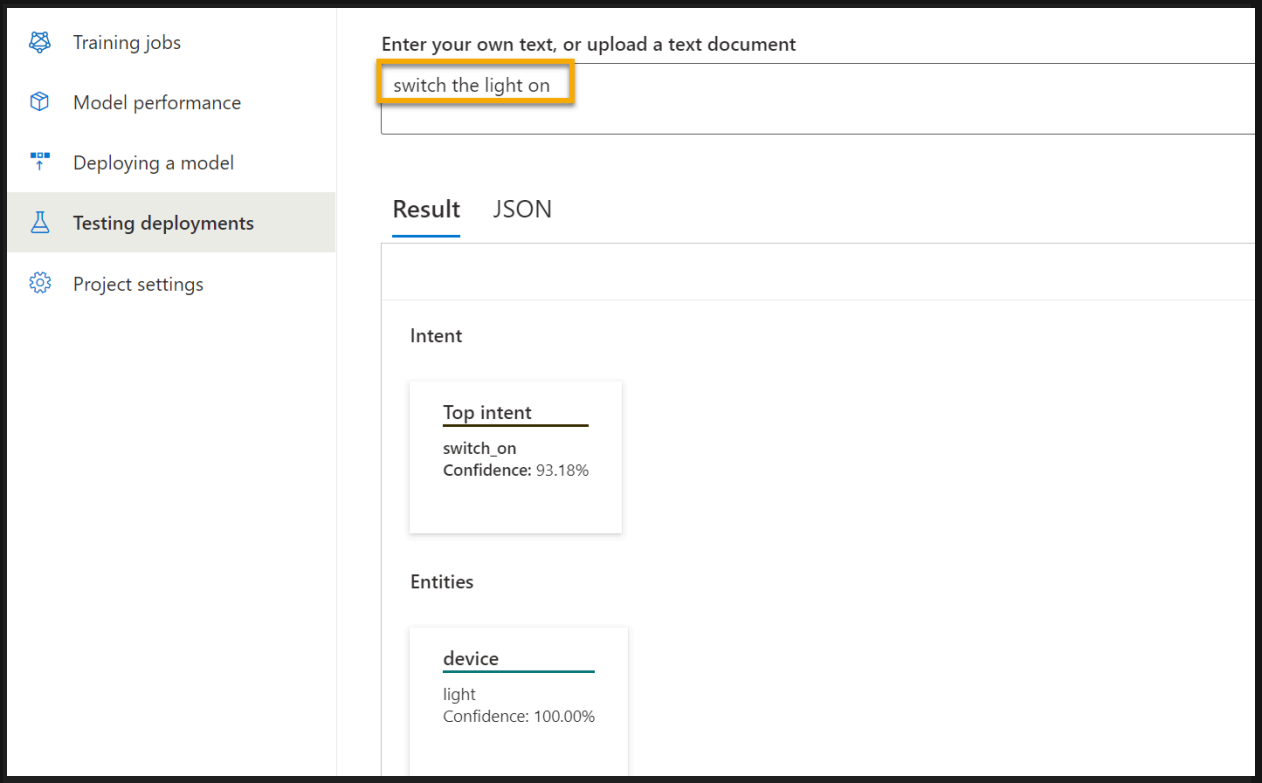
You can easily create a question answering solution on Microsoft Azure using Azure AI Language service. Azure AI Language includes a custom question answering feature that enables you to create a knowledge base of question and answer pairs that can be queried using natural language input.



Conversational language understanding

Azure AI Language service supports conversational language understanding (CLU). You can use CLU to build language models that interpret the meaning of phrases in a conversational setting. Conversational language understanding describes a set of features that can be used to build an end-to-end conversational application. In particular, the features enable you to customize natural language understanding models to predict the overall intention of an incoming phrase and extract important information from it.

One example of a CLU application is one that's able to turn devices on and off based on speech. The application is able to take in audio input such as, "Turn the light off", and understand an action it needs to take, such as turning a light off. Many types of tasks involving command and control, end-to-end conversation, and enterprise support can be completed with Azure AI Language's CLU feature.



Note

In modern AI solutions, multiple AI capabilities are often working together, evolving from, or building off of one another. These conversational AI capabilities may look similar to what generative AI capabilities look like today. Generative AI uses NLP as a foundation but extends beyond it by creating new content.

Next, let's look at Azure AI Translator's capabilities.

Azure AI Translator capabilities

Early attempts at machine translation applied literal translations. A literal translation is where each word is translated to the corresponding word in the target language. This approach presents some issues. For one case, there may not be an equivalent word in the target language. Another case is where literal translation can change the meaning of the phrase or not get the context correct.

Artificial intelligence systems must be able to understand, not only the words, but also the semantic context in which they're used. In this way, the service can return a more accurate translation of the input phrase or phrases. The grammar rules, formal versus informal, and colloquialisms all need to be considered.

Azure AI Translator supports text-to-text translation between more than 130 languages. When using Azure AI Translator, you can specify one from language with multiple to

languages, enabling you to simultaneously translate a source document into multiple languages.

Using Azure AI Translator

Azure AI Translator includes the following capabilities:

- Text translation - used for quick and accurate text translation in real time across all supported languages.
- Document translation - used to translate multiple documents across all supported languages while preserving original document structure.
- Custom translation - used to enable enterprises, app developers, and language service providers to build customized neural machine translation (NMT) systems.

You can use *Azure AI Translator* in Azure AI Foundry, a unified platform for enterprise AI operations, model builders, and application development. The service is also available for use in Microsoft Translator Pro a mobile application, designed specifically for enterprises, that enables seamless real-time speech-to-speech translation.

Next, let's learn how to get started with language capabilities in Azure AI Foundry portal.

Get started in Azure AI Foundry

Azure AI Language and Azure AI Translator provide the building blocks for incorporating language capabilities into applications. As one of many Azure AI services, you can create solutions in several ways including:

- The Azure AI Foundry portal
- A software development kit (SDK) or REST API

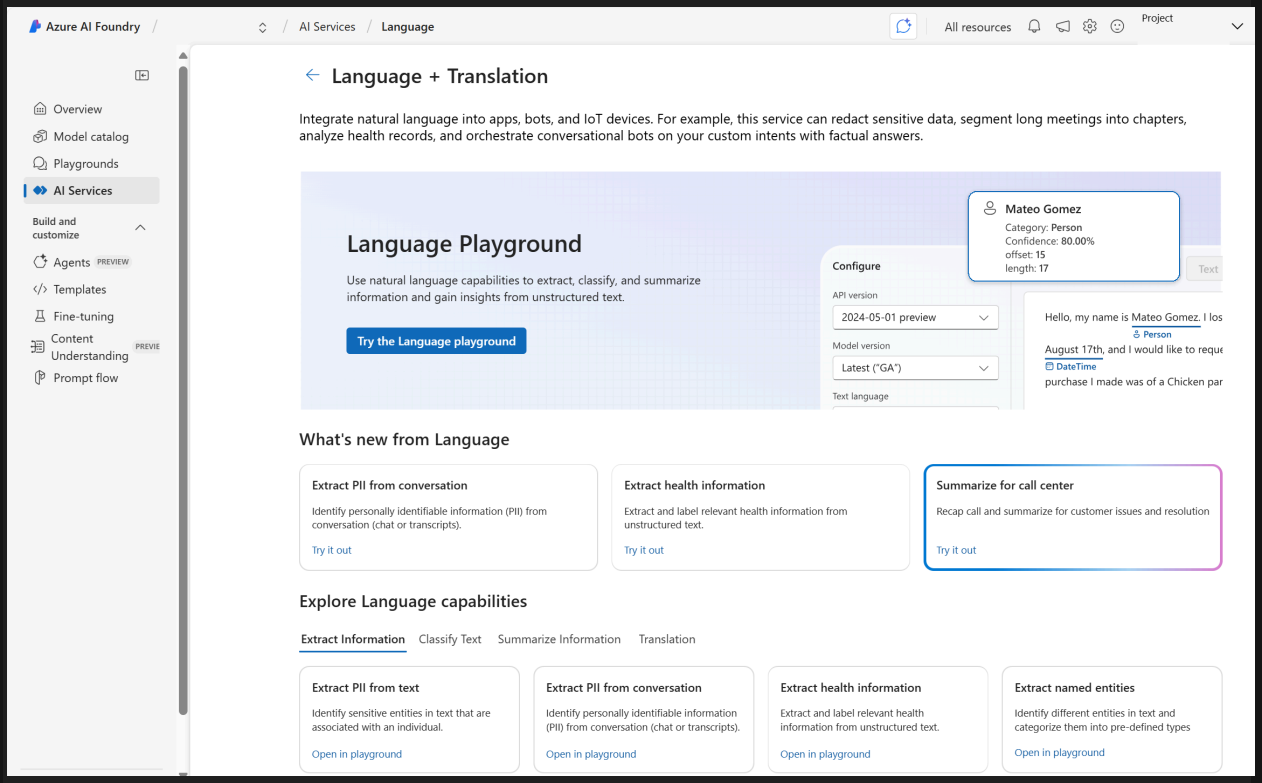
To use Azure AI Language or Azure AI Translator in an application, you must provision an appropriate resource in your Azure subscription. You can choose either a single-service resource or a multi-service resource.

- A Language resource - choose if you only plan to use Azure AI Language services, or if you want to manage access and billing for the resource separately from other services.
- A Translator resource - choose if you want to manage access and billing for each service individually.
- An Azure AI services resource - choose if you plan to use Azure AI Language in combination with other Azure AI services, and you want to manage access and billing for these services together.

Note

There are several ways to create resources with Azure. You can use a user interface to create resources or write a script. Both the Azure portal and Azure AI Foundry portal provide user interfaces for resource creation. Choose the Azure AI Foundry portal when you also want to see examples of Azure AI services in action.

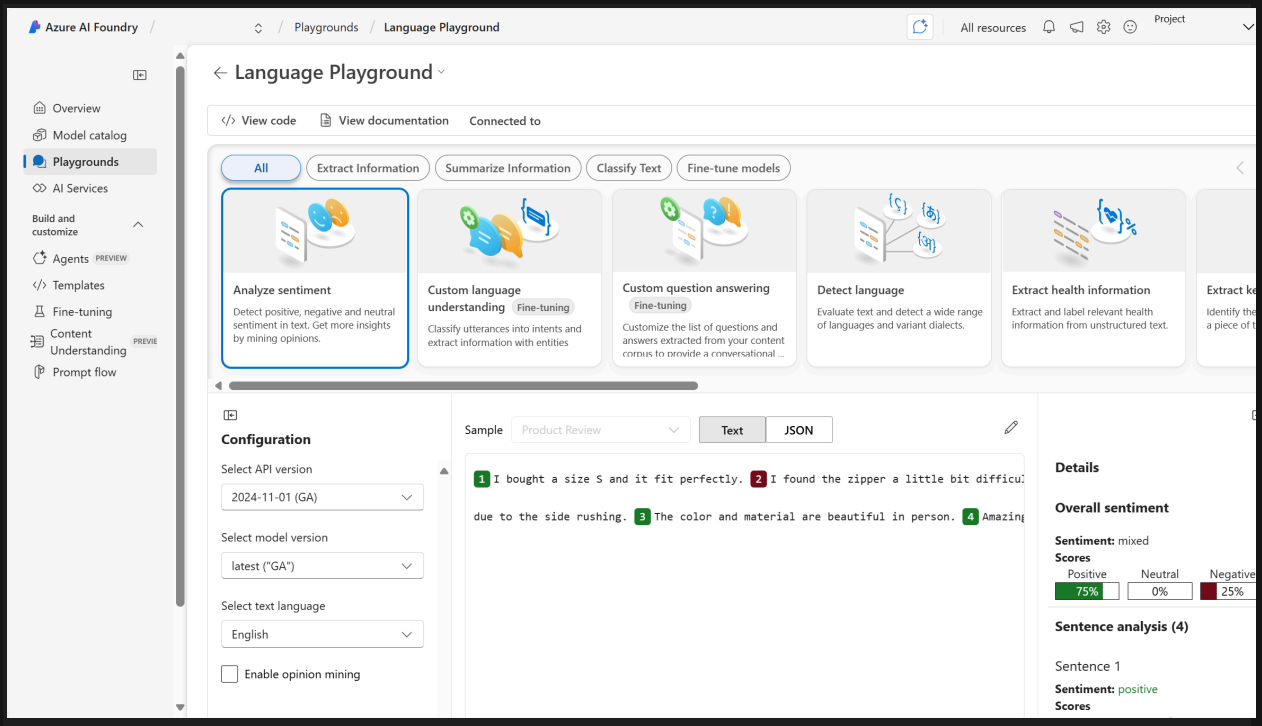
Get started in Azure AI Foundry portal

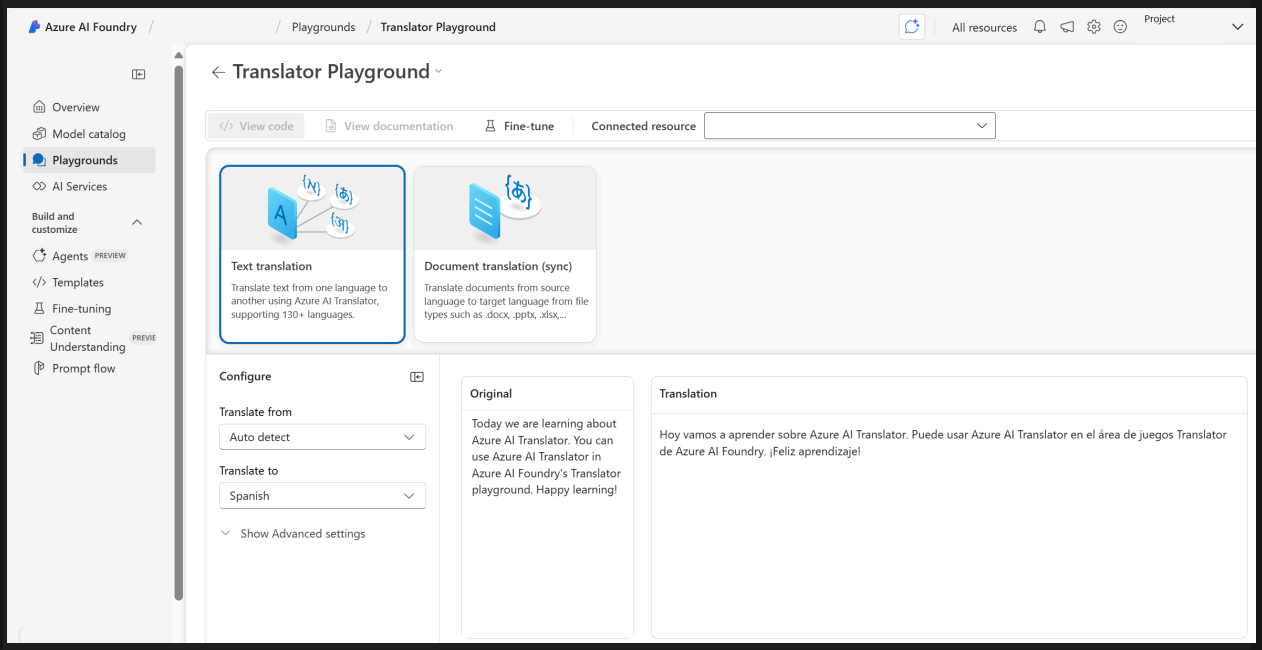


Azure AI Foundry provides a unified platform for enterprise AI operations, model builders, and application development. Azure AI Foundry portal provides a user interface based around hubs and projects. To use any of the Azure AI services, including Azure AI Language or Azure AI Translator, you create a project in Azure AI Foundry, which will also create an Azure AI services resource for you.

Projects in Azure AI Foundry help you organize your work and resources effectively. Projects act as containers for datasets, models, and other resources, making it easier to manage and collaborate on AI solutions.

Within Azure AI Foundry portal, you have the ability to try out service features in a playground setting. Azure AI Foundry portal provides a language playground and a translator playground.





Next let's try out language capabilities in Azure AI Foundry portal.

Introduction to AI speech concepts

Introduction

Speech is one of the most natural ways humans communicate, and bringing speech capabilities to AI applications creates more intuitive, accessible, and engaging user experiences. Whether you're building a voice assistant, creating accessible applications, or developing conversational AI agents, understanding speech technologies is essential for modern AI solutions.

In this module, you'll explore the two fundamental speech capabilities that power voice-enabled applications: speech recognition (converting spoken words to text) and speech synthesis (converting text to natural-sounding speech). You'll discover how these technologies work together to create seamless voice interactions and learn about the real-world scenarios where speech can transform user experiences.

Speech-enabled solutions

Speech capabilities transform how users interact with AI applications and agents. Speech recognition converts spoken words into text, while speech synthesis generates natural-sounding audio from text. Together, these technologies enable hands-free operation, improve accessibility, and create more natural conversational experiences.

Integrating speech into your AI solutions helps you:

- Expand accessibility: Serve users with visual impairments or mobility challenges.
- Increase productivity: Enable multitasking by removing the need for keyboards and screens.
- Enhance user experience: Create natural conversations that feel more human and engaging.
- Reach global audiences: Support multiple languages and regional dialects.

Common speech recognition scenarios

Speech recognition, also called speech-to-text, listens to audio input and transcribes it into written text. This capability powers a wide range of business and consumer applications.

Customer service and support

Service centers use speech recognition to:

- Transcribe customer calls in real time for agent reference and quality assurance.
- Route callers to the right department based on what they say.

- Analyze call sentiment and identify common customer issues.
- Generate searchable call records for compliance and training.

Business value: Reduces manual note-taking, improves response accuracy, and captures insights that improve service quality.

Voice-activated assistants and agents

Virtual assistants and AI agents rely on speech recognition to:

- Accept voice commands for hands-free control of devices and applications.
- Answer questions using natural language understanding.
- Complete tasks like setting reminders, sending messages, or searching information.
- Control smart home devices, automotive systems, and wearable technology.

Business value: Increases user engagement, simplifies complex workflows, and enables operation in situations where screens aren't practical.

Meeting and interview transcription

Organizations transcribe conversations to:

- Create searchable meeting notes and action item lists.
- Provide real-time captions for participants who are deaf or hard of hearing.
- Generate summaries of interviews, focus groups, and research sessions.
- Extract key discussion points for documentation and follow-up.

Business value: Saves hours of manual transcription work, ensures accurate records, and makes spoken content accessible to everyone.

Healthcare documentation

Clinical professionals use speech recognition to:

- Dictate patient notes directly into electronic health records.
- Update treatment plans without interrupting patient care.
- Reduce administrative burden and prevent physician burnout.
- Improve documentation accuracy by capturing details in the moment.

Business value: Increases time available for patient care, improves record completeness, and reduces documentation errors.

Common speech synthesis scenarios

Speech synthesis, also called text-to-speech, converts written text into spoken audio. This technology creates voices for applications that need to communicate information audibly.

Conversational AI and chatbots

AI agents use speech synthesis to:

- Respond to users with natural-sounding voices instead of requiring them to read text.
- Create personalized interactions by adjusting tone, pace, and speaking style.
- Handle customer inquiries through voice channels like phone systems.
- Provide consistent brand experiences across voice and text interfaces.

Business value: Makes AI agents more approachable, reduces customer effort, and extends service availability to voice-only channels.

Accessibility and content consumption

Applications generate audio to:

- Read web content, articles, and documents aloud for users with visual impairments.
- Support users with reading disabilities like dyslexia.
- Enable content consumption while driving, exercising, or performing other tasks.
- Provide audio alternatives for text-heavy interfaces.

Business value: Expands your audience reach, demonstrates commitment to inclusion, and improves user satisfaction.

Notifications and alerts

Systems use speech synthesis to:

- Announce important alerts, reminders, and status updates.
- Provide navigation instructions in mapping and GPS applications.
- Deliver time-sensitive information without requiring users to look at screens.
- Communicate system status in industrial and operational environments.

Business value: Ensures critical information reaches users even when visual attention isn't available, improving safety and responsiveness.

E-learning and training

Educational platforms use speech synthesis to:

- Create narrated lessons and course content without recording studios.
- Provide pronunciation examples for language learning.
- Generate audio versions of written materials for different learning preferences.
- Scale content production across multiple languages.

Business value: Reduces content creation costs, supports diverse learning styles, and accelerates course development timelines.

Entertainment and media

Content creators use speech synthesis to:

- Generate character voices for games and interactive experiences.
- Produce podcast drafts and audiobook prototypes.
- Create voiceovers for videos and presentations.
- Personalize audio content based on user preferences.

Business value: Lowers production costs, enables rapid prototyping, and creates customized experiences at scale.

Combining speech recognition and synthesis

The most powerful speech-enabled applications combine both capabilities to create conversational experiences:

- Voice-driven customer service: Agents listen to customer questions (recognition), process the request, and respond with helpful answers (synthesis).
- Interactive voice response (IVR) systems: Callers speak their needs, and the system guides them through options using natural dialogue.
- Language learning applications: Students speak practice phrases (recognition), and the system provides feedback and corrections (synthesis).
- Voice-controlled vehicles: Drivers give commands hands-free (recognition), and the system confirms actions and provides updates (synthesis).

These combined scenarios create fluid, two-way conversations that feel natural and reduce the friction users experience with traditional interfaces.

Tip

Start with a single speech capability focused on your highest-value scenario. Prove the concept works before expanding to more complex conversational flows.

Key considerations before implementing speech

Before you add speech capabilities to your application, evaluate these factors:

- Audio quality requirements: Background noise, microphone quality, and network bandwidth affect speech recognition accuracy.
- Language and dialect support: Verify that your target languages and regional variations are supported.
- Privacy and compliance: Understand how audio data is processed, stored, and protected to meet regulatory requirements.
- Latency expectations: Real-time conversations require low-latency processing, while batch transcription can tolerate delays.
- Accessibility standards: Ensure your speech implementation meets WCAG guidelines and doesn't create barriers for some users.

Important

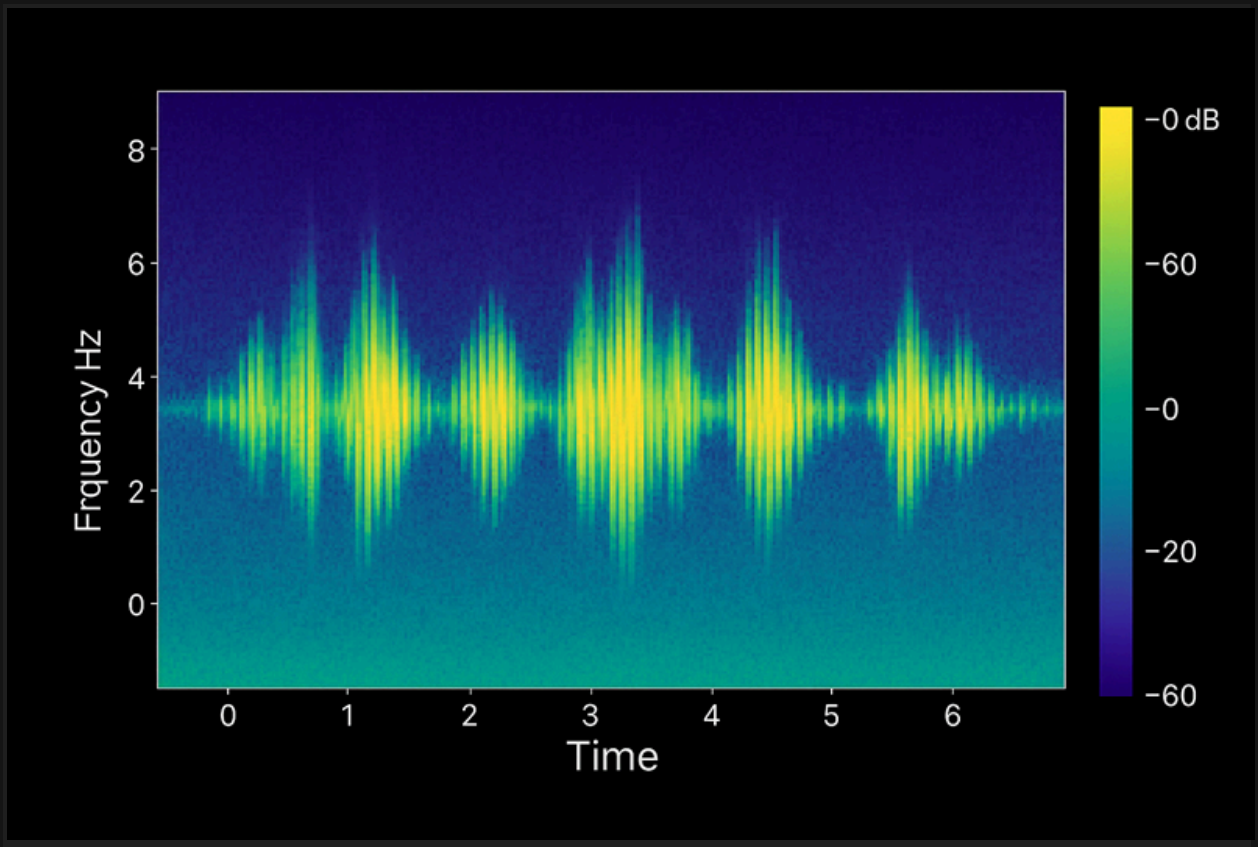
Always provide alternative input and output methods. Some users may prefer or require text-based interfaces even when speech is available.

Speech recognition

Speech recognition, also called speech-to-text, enables applications to convert spoken language into written text. The journey from sound wave to text involves six coordinated stages: capturing audio, preparing features, modeling acoustic patterns, applying language rules, decoding the most likely words, and refining the final output.

Audio capture: Convert analog audio to digital

Speech recognition begins when a microphone converts sound waves into a digital signal. The system samples the analog audio thousands of times per second—typically 16,000 samples per second (16 kHz) for speech applications—and stores each measurement as a numeric value.



Note

Why sampling rate matters:

- Higher rates (like 44.1 kHz for music) capture more detail but require more processing.
- Speech recognition balances clarity and efficiency at 8 kHz to 16 kHz.
- Background noise, microphone quality, and distance from the speaker directly impact downstream accuracy.

Before moving to the next stage, the system often applies basic filters to remove hums, clicks, or other background noise that could confuse the model.

Pre-processing: Extract meaningful features

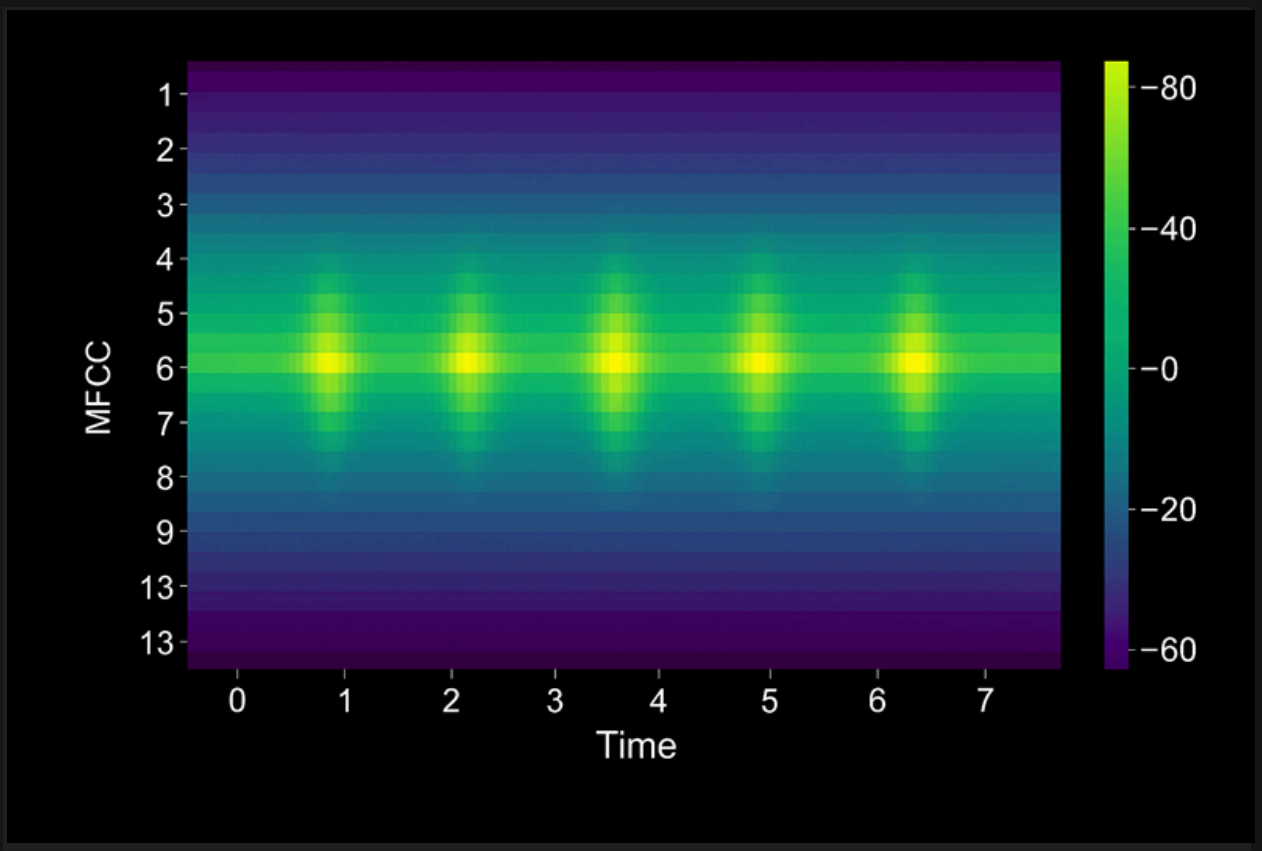
Raw audio samples contain too much information for efficient pattern recognition. Pre-processing transforms the waveform into a compact representation that highlights speech characteristics while discarding irrelevant details like absolute volume.

Mel-Frequency Cepstral Coefficients (MFCCs)

MFCC is the most common feature extraction technique in speech recognition. It mimics how the human ear perceives sound by emphasizing frequencies where speech energy concentrates and compressing less important ranges.

How MFCC works:

1. Divide audio into frames: Split the signal into overlapping 20–30 millisecond windows.
2. Apply Fourier transform: Convert each frame from time domain to frequency domain, revealing which pitches are present.
3. Map to Mel scale: Adjust frequency bins to match human hearing sensitivity—we distinguish low pitches better than high ones.
4. Extract coefficients: Compute a small set of numbers (often 13 coefficients) that summarize the spectral shape of each frame.



The result is a sequence of feature vectors—one per frame—that captures what the audio sounds like without storing every sample. These vectors become the input for acoustic modeling.

The vectors are extracted column-wise, with each vector representing the 13 MFCC feature coefficient values for each time-frame:

```
Frame 1: [ -113.2,  45.3,  12.1,  -3.4,  7.8,  ... ]  # 13 coefficients

Frame 2: [ -112.8,  44.7,  11.8,  -3.1,  7.5,  ... ]

Frame 3: [ -110.5,  43.9,  11.5,  -2.9,  7.3,  ... ]
```

Acoustic modeling: Recognize phonemes

Acoustic models learn the relationship between audio features and phonemes—the smallest units of sound that distinguish words. English uses about 44 phonemes; for example, the word "cat" comprises three phonemes: /k/, /æ/, and /t/.

From features to phonemes

Modern acoustic models use transformer architectures, a type of deep learning network that excels at sequence tasks. The transformer processes the MFCC feature vectors and predicts which phoneme is most likely at each moment in time.

Transformer models achieve effective phoneme prediction through:

- Attention mechanism: The model examines surrounding frames to resolve ambiguity. For example, the phoneme /t/ sounds different at the start of "top" versus the end of "bat."
- Parallel processing: Unlike older recurrent models, transformers analyze multiple frames simultaneously, improving speed and accuracy.
- Contextualized predictions: The network learns that certain phoneme sequences occur frequently in natural speech.

The output of acoustic modeling is a probability distribution over phonemes for each audio frame. For instance, frame 42 might show 80% confidence for /æ/, 15% for /ɛ/, and 5% for other phonemes.

Note

Phonemes are language-specific. A model trained on English phonemes can't recognize Mandarin tones without retraining.

Language modeling: Predict word sequences

Phoneme predictions alone don't guarantee accurate transcription. The acoustic model might confuse "their" and "there" because they share identical phonemes. Language models resolve ambiguity by applying knowledge of vocabulary, grammar, and common word patterns. Some ways in which the model guides word sequence prediction include:

- Statistical patterns: The model knows "The weather is nice" appears more often in training data than "The whether is nice."
- Context awareness: After hearing "I need to," the model expects verbs like "go" or "finish," not nouns like "table."
- Domain adaptation: Custom language models trained on medical or legal terminology improve accuracy for specialized scenarios.

Decoding: Select the best text hypothesis

Decoding algorithms search through millions of possible word sequences to find the transcription that best matches both acoustic and language model predictions. This stage balances two competing goals: staying faithful to the audio signal while producing readable, grammatically correct text.

Beam search decoding:

The most common technique, *beam search*, maintains a shortlist (the "beam") of top-scoring partial transcriptions as it processes each audio frame. At every step, it extends each hypothesis with the next most likely word, prunes low-scoring paths, and keeps only the best candidates.

For a three-second utterance, the decoder might evaluate thousands of hypotheses before selecting "Please send the report by Friday" over alternatives like "Please sent the report buy Friday."

Caution

Decoding is computationally intensive. Real-time applications balance accuracy and latency by limiting beam width and hypothesis depth.

Post-processing: Refine the output

The decoder produces raw text that often requires cleanup before presentation. Post-processing applies formatting rules and corrections to improve readability and accuracy.

Common post-processing tasks:

- Capitalization: Convert "hello my name is sam" to "Hello my name is Sam."
- Punctuation restoration: Add periods, commas, and question marks based on prosody and grammar.
- Number formatting: Change "one thousand twenty three" to "1,023."
- Profanity filtering: Mask or remove inappropriate words when required by policy.
- Inverse text normalization: Convert spoken forms like "three p m" to "3 PM."
- Confidence scoring: Flag low-confidence words for human review in critical applications like medical transcription.

Azure AI Speech returns the final transcription along with metadata like word-level timestamps and confidence scores, enabling your application to highlight uncertain segments or trigger fallback behaviors.

How the pipeline works together

Each stage builds on the previous one:

1. Audio capture provides the raw signal.
2. Pre-processing extracts MFCC features that highlight speech patterns.
3. Acoustic modeling predicts phoneme probabilities using transformer networks.
4. Language modeling applies vocabulary and grammar knowledge.
5. Decoding searches for the best word sequence.
6. Post-processing formats the text for human readers.

By separating concerns, modern speech recognition systems achieve high accuracy across languages, accents, and acoustic conditions. When transcription quality falls short, you can often trace the issue to one stage—poor audio capture, insufficient language model training, or overly aggressive post-processing—and adjust accordingly.

Speech synthesis

Speech synthesis—also called text-to-speech (TTS)—converts written text into spoken audio. You encounter speech synthesis when virtual assistants read notifications, navigation apps announce directions, or accessibility tools help users consume written content audibly.

Speech synthesis systems process text through four distinct stages. Each stage transforms the input incrementally, building toward a final audio waveform that sounds natural and intelligible.

Text normalization: Standardize the text

Text normalization prepares raw text for pronunciation by expanding abbreviations, numbers, and symbols into spoken forms.

Consider the sentence: "*Dr. Smith ordered 3 items for $25.50 on 12/15/2023.*"

A normalization system converts it to: "Doctor Smith ordered three items for twenty-five dollars and fifty cents on December fifteenth, two thousand twenty-three."

Common normalization tasks include:

- Expanding abbreviations ("Dr." becomes "Doctor", "Inc." becomes "Incorporated")
- Converting numbers to words ("3" becomes "three", "25.50" becomes "twenty-five point five zero")
- Handling dates and times ("12/15/2023" becomes "December fifteenth, two thousand twenty-three")
- Processing symbols and special characters ("$" becomes "dollars", "@" becomes "at")
- Resolving homographs based on context ("read" as present tense versus past tense)

Text normalization prevents the system from attempting to pronounce raw symbols or digits, which would produce unnatural or incomprehensible output.

Tip

Different domains require specialized normalization rules. Medical text handles drug names and dosages differently than financial text handles currency and percentages.

Linguistic analysis: Map text to phonemes

Linguistic analysis breaks normalized text into *phonemes* (the smallest units of sound) and determines how to pronounce each word. The linguistic analysis stage:

1. Segments text into words and syllables.
2. Looks up word pronunciations in lexicons (pronunciation dictionaries).
3. Applies G2P rules or neural models to handle unknown words.
4. Marks syllable boundaries and identifies stressed syllables.
5. Determines phonetic context for adjacent sounds.

Grapheme-to-phoneme conversion

Grapheme-to-phoneme (G2P) conversion maps written letters (*graphemes*) to pronunciation sounds (*phonemes*). English spelling doesn't reliably indicate pronunciation, so G2P systems use both rules and learned patterns.

For example:

- The word "though" converts to /θoʊ/
- The word "through" converts to /θruː/
- The word "cough" converts to /kɔːf/

Each word contains the letters "ough", but the pronunciation differs dramatically.

Modern G2P systems use neural networks trained on pronunciation dictionaries. These models learn patterns between spelling and sound, handling uncommon words, proper names, and regional variations more gracefully than rule-based systems.

When determining phonemes, linguistic analysis often uses a *transformer* model to help consider *context*. For example, the word "*read*" is pronounced differently in "I *read* books" (present tense: /riːd/) versus "I *read* that book yesterday" (past tense: /rɛd/).

Prosody generation: Determine pronunciation

Prosody refers to the rhythm, stress, and intonation patterns that make speech sound natural. Prosody generation determines how to say words, not just which sounds to produce.

Elements of prosody

Prosody encompasses several vocal characteristics:

- Pitch contours: Rising or falling pitch patterns that signal questions versus statements
- Duration: How long to hold each sound, creating emphasis or natural rhythm
- Intensity: Volume variations that highlight important words
- Pauses: Breaks between phrases or sentences that aid comprehension
- Stress patterns: Which syllables receive emphasis within words and sentences

Prosody has a significant effect on how spoken text is interpreted. For example, consider how the following sentence changes meaning depending on which syllable or word is emphasized:

- "*I* never said he ate the cake."
- "I never said *he* ate the cake."

- "I never said he *ate* the cake."
- "I never said he ate the *cake*."

Transformer-based prosody prediction

Modern speech synthesis systems use transformer neural networks to predict prosody. Transformers excel at understanding context across entire sentences, not just adjacent words.

The prosody generation process:

1. Input encoding: The transformer receives the phoneme sequence with linguistic features (punctuation, part of speech, sentence structure)
2. Contextual analysis: Self-attention mechanisms identify relationships between words (for example, which noun a pronoun references, where sentence boundaries fall)
3. Prosody prediction: The model outputs predicted values for pitch, duration, and energy at each phoneme
4. Style factors: The system considers speaking style (neutral, expressive, conversational) and speaker characteristics

Transformers predict prosody by learning from thousands of hours of recorded speech paired with transcripts. The model discovers patterns: questions rise in pitch at the end, commas signal brief pauses, emphasized words lengthen slightly, and sentence-final words often drop in pitch.

Factors influencing prosody choices:

- Syntax: Clause boundaries indicate where to pause
- Semantics: Important concepts receive emphasis
- Discourse context: Contrasting information or answers to questions may carry extra stress
- Speaker identity: Each voice has characteristic pitch range and speaking rate
- Emotional tone: Excitement, concern, or neutrality shape prosodic patterns

The prosody predictions create a target specification: "Produce the phoneme /æ/ at 180 Hz for 80 milliseconds with moderate intensity, then pause for 200 milliseconds."

Important

Prosody dramatically affects naturalness. Robotic-sounding speech often results from flat, monotone prosody—not from imperfect phoneme pronunciation.

Speech synthesis: Generate audio

Speech synthesis generates the final audio waveform based on the phoneme sequence and prosody specifications.

Waveform generation approaches

Modern systems use neural vocoders—deep learning models that generate audio samples directly. Popular vocoder architectures include WaveNet, WaveGlow, and HiFi-GAN.

The synthesis process:

1. Acoustic feature generation: An acoustic model (often a transformer) converts phonemes and prosody targets into mel-spectrograms—visual representations of sound frequencies over time
2. Vocoding: The neural vocoder converts mel-spectrograms into raw audio waveforms (sequences of amplitude values at 16,000-48,000 samples per second)
3. Post-processing: The system applies filtering, normalization, or audio effects to match target output specifications

Note

What makes neural vocoders effective:

- High fidelity: Generate audio quality approaching studio recordings
- Naturalness: Capture subtle vocal characteristics like breathiness and voice quality
- Efficiency: Real-time generation on modern hardware (important for interactive applications)
- Flexibility: Adapt to different speakers, languages, and speaking styles

The vocoder essentially performs the inverse of what automatic speech recognition does—while speech recognition converts audio into text, the vocoder converts linguistic representations into audio.

The complete pipeline in action

When you request speech synthesis for "Dr. Chen's appointment is at 3:00 PM":

1. Text normalization expands it to "Doctor Chen's appointment is at three o'clock P M"
2. Linguistic analysis converts it to phonemes: /ˈdɑktər ˈtʃɛnz əˈpɔɪntmənt ɪz æt θri əˈklɑk pi ɛm/
3. Prosody generation predicts pitch rising slightly on "appointment", a pause after "is", and emphasis on "three"
4. Speech synthesis generates an audio waveform matching those specifications

The entire process typically completes in under one second on modern hardware.

Get started with speech in Azure

Introduction

AI speech capabilities enable us to manage home and auto systems with voice instructions, get answers from computers for spoken questions, generate captions from audio, and much more.

To enable this kind of interaction, the AI system must support at least two capabilities:

- Speech recognition - the ability to detect and interpret spoken input
- Speech synthesis - the ability to generate spoken output

Azure AI Speech provides speech to text, text to speech, and speech translation capabilities through speech recognition and synthesis. You can use prebuilt and custom Speech service models for a variety of tasks, from transcribing audio to text with high accuracy, to identifying speakers in conversations, creating custom voices, and more. Next you'll learn how AI speech capabilities work.

Understand speech recognition and synthesis

Speech recognition takes the spoken word and converts it into data that can be processed - often by transcribing it into text. The spoken words can be in the form of a recorded voice in an audio file, or live audio from a microphone. Speech patterns are analyzed in the audio to determine recognizable patterns that are mapped to words. To accomplish this, the software typically uses multiple models, including:

- An *acoustic* model that converts the audio signal into phonemes (representations of specific sounds).
- A *language* model that maps phonemes to words, usually using a statistical algorithm that predicts the most probable sequence of words based on the phonemes.

The recognized words are typically converted to text, which you can use for various purposes, such as:

- Providing closed captions for recorded or live videos
- Creating a transcript of a phone call or meeting
- Automated note dictation
- Determining intended user input for further processing

Speech synthesis is concerned with vocalizing data, usually by converting text to speech. A speech synthesis solution typically requires the following information:

- The text to be spoken
- The voice to be used to vocalize the speech

To synthesize speech, the system typically *tokenizes* the text to break it down into individual words, and assigns phonetic sounds to each word. It then breaks the phonetic transcription into *prosodic* units (such as phrases, clauses, or sentences) to create phonemes that will be converted to audio format. These phonemes are then synthesized as audio and can be assigned a particular voice, speaking rate, pitch, and volume.

You can use the output of speech synthesis for many purposes, including:

- Generating spoken responses to user input
- Creating voice menus for phone systems
- Reading email or text messages aloud in hands-free scenarios
- Broadcasting announcements in public locations, such as railway stations or airports
-

Get started with speech on Azure

Microsoft Azure offers speech recognition and synthesis capabilities through Azure AI Speech service, which supports many capabilities, including:

- Speech to text
- Text to speech
- Speech translation

Speech to text

You can use Azure AI Speech to text API to perform real-time or batch transcription of audio into a text format. The audio source for transcription can be a real-time audio stream from a microphone or an audio file.

Azure AI's Speech to text API is based on Microsoft's Universal Language Model. The data for the model is Microsoft-owned and deployed to Azure. The model is optimized for two scenarios, conversational and dictation. You can also create and train your own custom models including acoustics, language, and pronunciation if the prebuilt models from Microsoft don't provide what you need.

Real-time transcription: Real-time speech to text allows you to transcribe audio streams to text. You can use real-time transcription for presentations, demos, or any other scenario where a person is speaking.

In order for real-time transcription to work, your application needs to be listening for incoming audio from a microphone, or other audio input source such as an audio file. Your application code streams the audio to the service, which returns the transcribed text.

Batch transcription: Not all speech to text scenarios are real time. You might have audio recordings stored on a file share, a remote server, or even on Azure storage. You can point to audio files with a shared access signature (SAS) URI and asynchronously receive transcription results.

Batch transcription should be run in an asynchronous manner because the batch jobs are scheduled on a *best-effort basis*. Normally a job starts executing within minutes of the request but there's no estimate for when a job changes into the running state.

Text to speech

The text to speech API enables you to convert text input to audible speech, which can either be played directly through a computer speaker or written to an audio file.

Speech synthesis voices: When you use the text to speech API, you can specify the voice to be used to vocalize the text. This capability offers you the flexibility to personalize your speech synthesis solution and give it a specific character.

The service includes multiple predefined voices with support for multiple languages and regional pronunciation, including *neural* voices that leverage *neural networks* to overcome common limitations in speech synthesis with regard to intonation, resulting in a more natural sounding voice. You can also develop custom voices and use them with the text to speech API.

Speech translation

Azure Speech Translation is a feature of the Azure AI Speech service. Azure Speech Translation enables real-time translation of spoken language by taking inputs of audio streams and returning text in a specified language. It works by first converting speech to text using automatic speech recognition (ASR), then translating the recognized text into one or more target languages using machine translation. The service supports a wide range of source and target languages and can deliver translations as text or synthesized speech. Developers can integrate this functionality into applications using REST APIs or SDKs. These applications work well in scenarios like multilingual meetings, live event captioning, or global customer support.

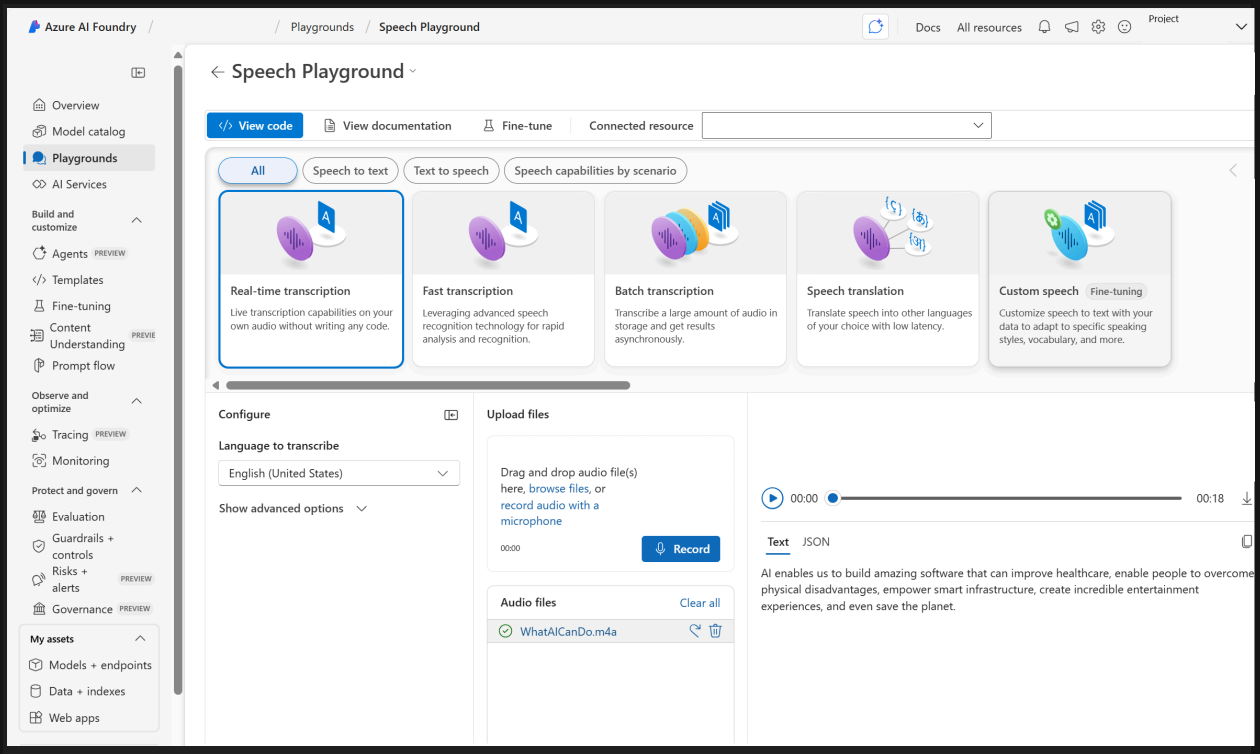
Use Azure AI Speech

Azure AI Speech is available for use through several tools and programming languages including:

- Studio interfaces
- Command Line Interface (CLI)
- REST APIs and Software Development Kits (SDKs)

Using studio interfaces

You can create Azure AI Speech projects using Azure AI Foundry portal's Speech Playground.



Azure resources for Azure AI Speech

To use Azure AI Speech in an application, you must create an appropriate resource in your Azure subscription. You can choose to create either of the following types of resource:

- A Speech resource - choose this resource type if you only plan to use Azure AI Speech, or if you want to manage access and billing for the resource separately from other services.
- An Azure AI services resource - choose this resource type if you plan to use Azure AI Speech in combination with other Azure AI services, and you want to manage access and billing for these services together.
-

Introduction to computer vision concepts

Introduction

Computer vision is one of the core areas of artificial intelligence (AI), and focuses on creating solutions that enable AI applications to process visual information.

Consider these scenarios:

- An autonomous vehicle needs to detect and respond to traffic and pedestrians.
- A store uses smart checkouts with cameras to determine the products in a customer's basket.
- A doorbell camera is used to detect people at your front door.

These use cases, and many others, rely on computer vision.

Of course, computers don't have biological eyes that work the way ours do, but they're capable of processing images; either from a live camera feed or from digital photographs or videos. This ability to process images is the key to creating software that can emulate human visual perception. In this module, we'll examine the building blocks that underlie modern computer vision solutions.

Computer vision tasks and techniques

The term "computer vision" refers to a range of tasks and techniques in which AI software processes visual input; typically from images, videos, or live camera streams. Computer vision is a well-established field of AI, and the techniques used to extract information from visual input have evolved significantly over the years.

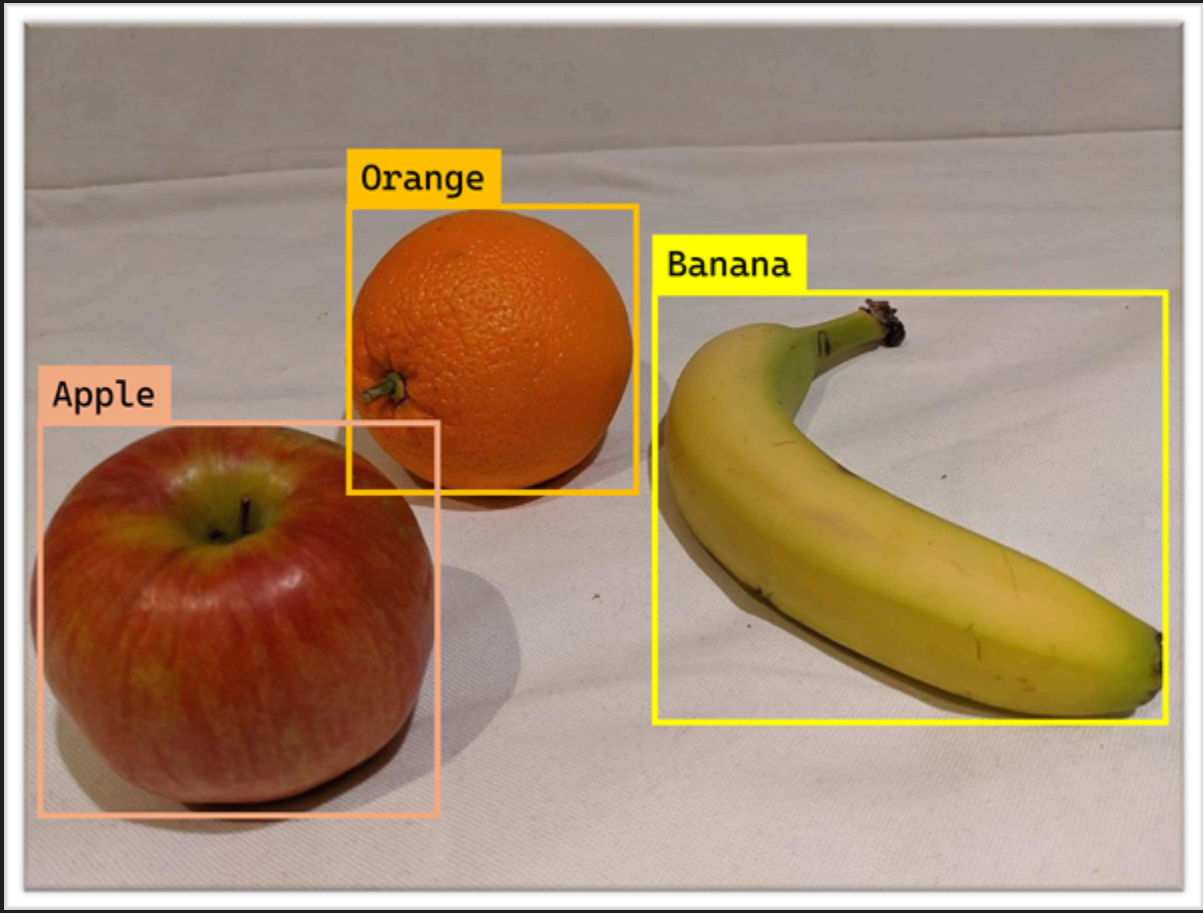
Image classification

One of the oldest computer vision solutions is a technique called *image classification*, in which a model that has been trained with a large number of images is used to predict a text label based on an image's contents.

For example, suppose a grocery store wants to implement smart checkout system that identifies produce automatically. For example, the customer could place fruits or vegetables on a scale at the checkout, and an AI application connected to a camera could automatically identify the types of produce (apple, orange, banana, and so on) and charge the appropriate amount based on its weight. For this solution to work, a model would need to be trained with a large volume of images, each labeled with the correct name. The result is a model that can use the visual features of an image to predict its main subject.

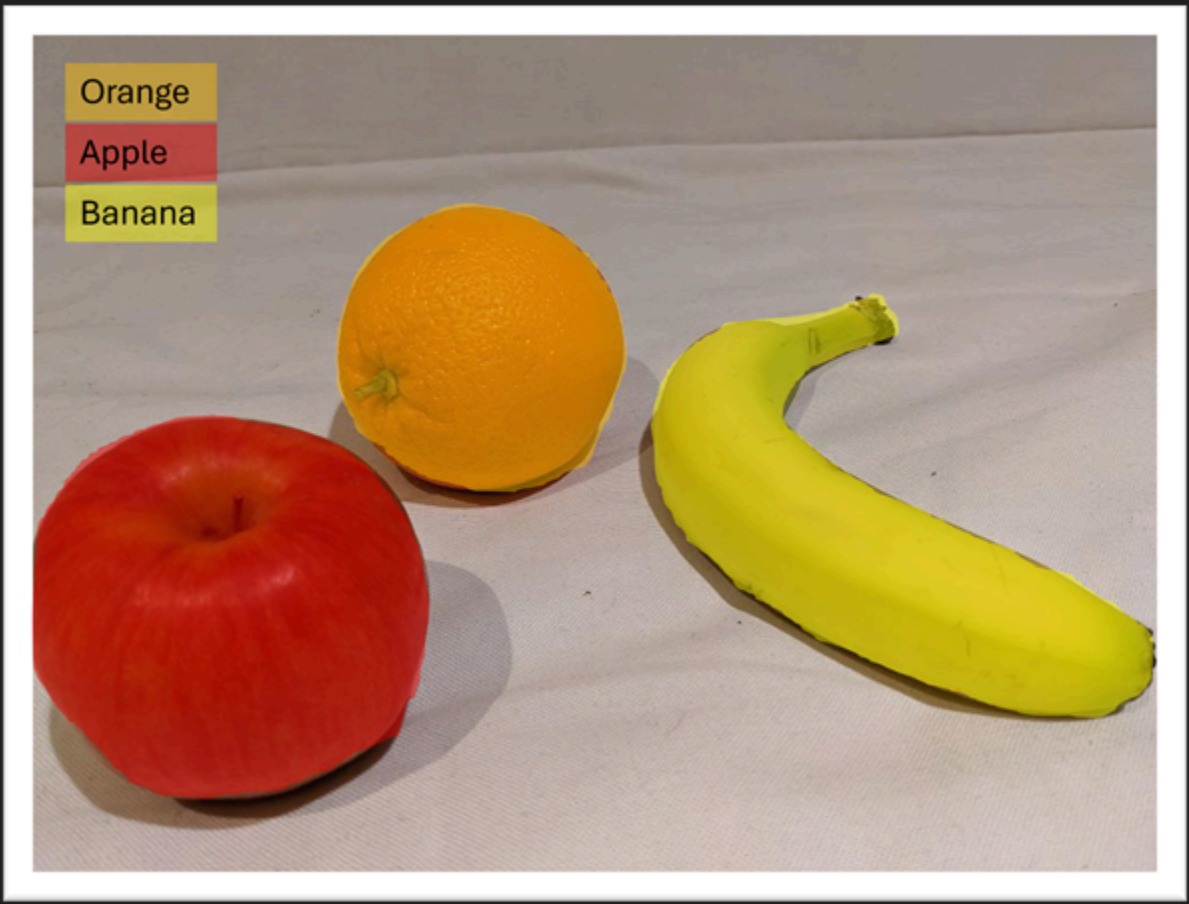
Object detection

Suppose the grocery store wants a more sophisticated system, in which the checkout can scan multiple items on the checkout and identify each of them. A common approach to this type of problem is called "object detection". Object detection models examine multiple regions in an image to find individual objects and their locations. The resulting prediction from the model includes which objects were detected, and the specific regions of the image in which they appear - indicated by the coordinates of the rectangular bounding box.



Semantic segmentation

Another, more sophisticated way to detect objects in an image, is called "semantic segmentation". In this approach, a model is trained to find objects, and classify individual pixels in the image based on the object to which they belong. The result of this process is a much more precise prediction of the location of objects in the image.



Contextual image analysis

The latest *multimodal* computer vision models are trained to find contextual relationships between objects in images and the text that describes them. The result is an ability to semantically interpret an image to determine what objects and activities it depicts; and generate appropriate descriptions or suggest relevant tags.

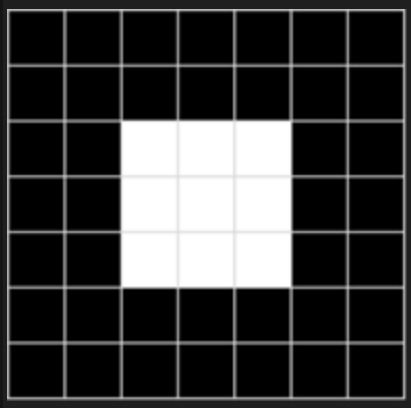
A person eating an apple.

Images and image processing

To a computer, an image is an array of numeric *pixel* values. For example, consider the following array:

```
0   0   0   0   0   0   0

 0   0   0   0   0   0   0

 0   0 255 255 255   0   0

 0   0 255 255 255   0   0

 0   0 255 255 255   0   0

 0   0   0   0   0   0   0

 0   0   0   0   0   0   0
```

The array consists of seven rows and seven columns, representing the pixel values for a 7x7 pixel image (which is known as the image's *resolution*). Each pixel has a value between 0 (black) and 255 (white); with values between these bounds representing shades of gray. The image represented by this array looks similar to the following (magnified) image:



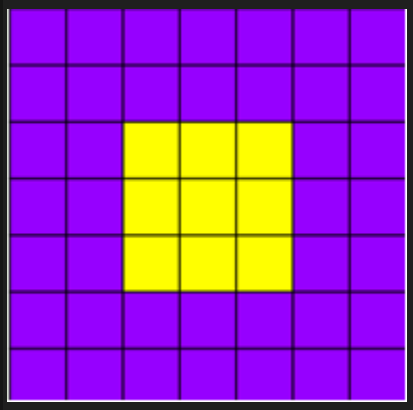
The array of pixel values for this image is two-dimensional (representing rows and columns, or *x* and *y* coordinates) and defines a single rectangle of pixel values. A single layer of pixel values like this represents a grayscale image. In reality, most digital images are multidimensional and consist of three layers (known as *channels*) that represent red, green, and blue (RGB) color hues. For example, we could represent a color image by defining three channels of pixel values that create the same square shape as the previous grayscale example:

```
Red:

 150  150  150  150  150  150  150

 150  150  150  150  150  150  150

 150  150  255  255  255  150  150

 150  150  255  255  255  150  150

 150  150  255  255  255  150  150

 150  150  150  150  150  150  150

 150  150  150  150  150  150  150


Green:

 0    0    0    0    0    0    0

 0    0    0    0    0    0    0
```

```
 0     0   255   255   255    0     0
 0     0   255   255   255    0     0
 0     0   255   255   255    0     0
 0     0     0     0     0    0     0
 0     0     0     0     0    0     0
```

```
Blue:
 255   255   255   255   255   255   255
 255   255   255   255   255   255   255
 255   255    0     0     0    255   255
 255   255    0     0     0    255   255
 255   255    0     0     0    255   255
 255   255   255   255   255   255   255
 255   255   255   255   255   255   255
```

Here's the resulting image:



The purple squares are represented by the combination:

```
Red: 150
Green: 0
```

```
Blue: 255
```

The yellow squares in the center are represented by the combination:

```
Red: 255

Green: 255

Blue: 0
```

Filters

A common way to perform image processing tasks is to apply *filters* that modify the pixel values of the image to create a visual effect. A filter is defined by one or more arrays of pixel values, called filter *kernels*. For example, you could define filter with a 3x3 kernel as shown in this example:

```
-1 -1 -1

-1  8 -1

-1 -1 -1
```

The kernel is then *convolved* across the image, calculating a weighted sum for each 3x3 patch of pixels and assigning the result to a new image. It's easier to understand how the filtering works by exploring a step-by-step example.

Let's start with the grayscale image we explored previously:

```
0   0   0   0   0   0   0

 0   0   0   0   0   0   0

 0   0  255 255 255  0   0

 0   0  255 255 255  0   0

 0   0  255 255 255  0   0

 0   0   0   0   0   0   0
```

```
 0   0   0   0   0   0   0
```

First, we apply the filter kernel to the top left patch of the image, multiplying each pixel value by the corresponding weight value in the kernel and adding the results:

```
(0 x -1) + (0 x -1) + (0 x -1) +

(0 x -1) + (0 x 8) + (0 x -1) +

(0 x -1) + (0 x -1) + (255 x -1) = -255
```

The result (-255) becomes the first value in a new array. Then we move the filter kernel along one pixel to the right and repeat the operation:

```
(0 x -1) + (0 x -1) + (0 x -1) +

(0 x -1) + (0 x 8) + (0 x -1) +

(0 x -1) + (255 x -1) + (255 x -1) = -510
```

Again, the result is added to the new array, which now contains two values:

```
-255  -510
```

The process is repeated until the filter has been convolved across the entire image, as shown in this animation:

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	255	255	255	0	0
0	0	255	255	255	0	0
0	0	255	255	255	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

The filter is convolved across the image, calculating a new array of values. Some of the values might be outside of the 0 to 255 pixel value range, so the values are adjusted to fit into that range. Because of the shape of the filter, the outside edge of pixels isn't calculated, so a padding value (usually 0) is applied. The resulting array represents a new image in which the filter has transformed the original image. In this case, the filter has had the effect of highlighting the *edges* of shapes in the image.

To see the effect of the filter more clearly, here's an example of the same filter applied to a real image:

Original Image	**Filtered Image**

Because the filter is convolved across the image, this kind of image manipulation is often referred to as *convolutional filtering*. The filter used in this example is a particular type of filter (called a *Laplace* filter) that highlights the edges on objects in an image. There are many other kinds of filter that you can use to create blurring, sharpening, color inversion, and other effects.

Convolutional neural networks

The ability to use filters to apply effects to images is useful in image processing tasks, such as you might perform with image editing software. However, the goal of computer vision is often to extract meaning, or at least actionable insights, from images; which requires the creation of machine learning models that are trained to recognize features based on large volumes of existing images.

Tip

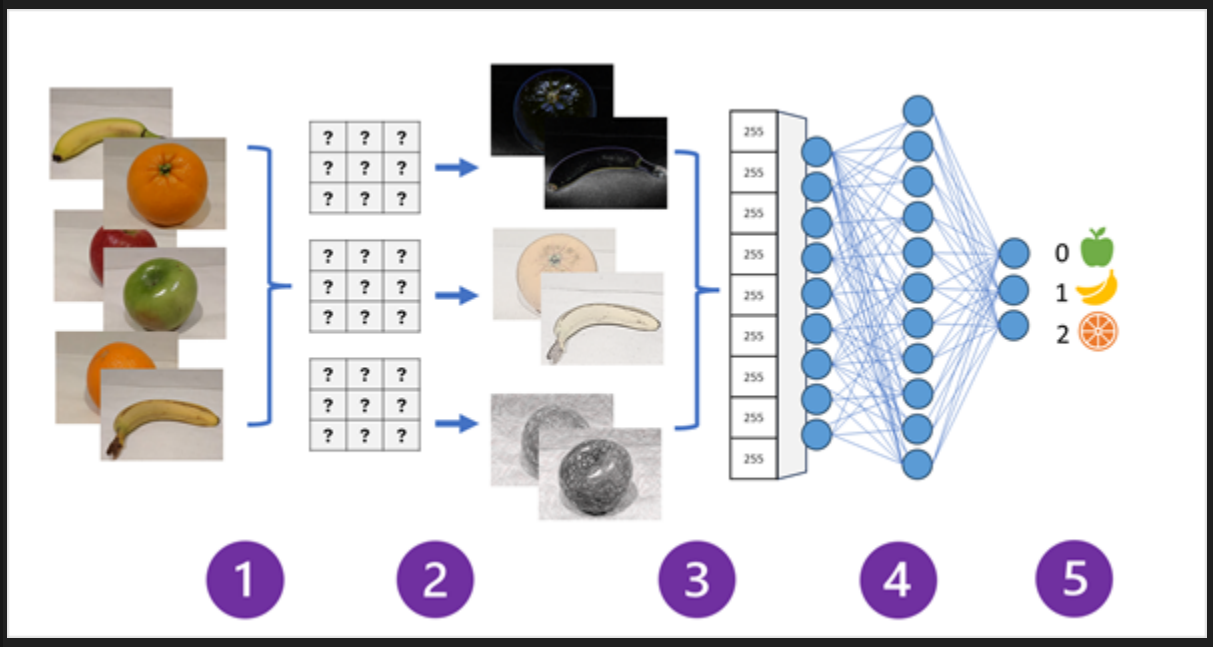
This unit assumes you are familiar with the fundamental principles of machine learning, and that you have conceptual knowledge of deep learning with neural networks. If you are new to machine learning, consider completing the Introduction to machine learning concepts module on Microsoft Learn.

One of the most common machine learning model architectures for computer vision is a *convolutional neural network* (CNN), a type of deep learning architecture. CNNs use filters to extract numeric feature maps from images, and then feed the feature values into a deep learning model to generate a label prediction. For example, in an *image classification* scenario, the label represents the main subject of the image (in other

words, what is this an image of?). You might train a CNN model with images of different kinds of fruit (such as apple, banana, and orange) so that the label that is predicted is the type of fruit in a given image.

During the *training* process for a CNN, filter kernels are initially defined using randomly generated weight values. Then, as the training process progresses, the models predictions are evaluated against known label values, and the filter weights are adjusted to improve accuracy. Eventually, the trained fruit image classification model uses the filter weights that best extract features that help identify different kinds of fruit.

The following diagram illustrates how a CNN for an image classification model works:



1. Images with known labels (for example, 0: apple, 1: banana, or 2: orange) are fed into the network to train the model.
2. One or more layers of filters is used to extract features from each image as it is fed through the network. The filter kernels start with randomly assigned weights and generate arrays of numeric values called *feature maps*. Additional layers may "pool" or "downsize" the feature maps to create smaller arrays that emphasize the key visual features extracted by the filters.
3. The feature maps are flattened into a single dimensional array of feature values.
4. The feature values are fed into a fully connected neural network.
5. The output layer of the neural network uses a *softmax* or similar function to produce a result that contains a probability value for each possible class, for example [0.2, 0.5, 0.3].

During training the output probabilities are compared to the actual class label - for example, an image of a banana (class 1) should have the value [0.0, 1.0, 0.0]. The

difference between the predicted and actual class scores is used to calculate the *loss* in the model, and the weights in the fully connected neural network and the filter kernels in the feature extraction layers are modified to reduce the loss.

The training process repeats over multiple *epochs* until an optimal set of weights has been learned. Then, the weights are saved and the model can be used to predict labels for new images for which the label is unknown.

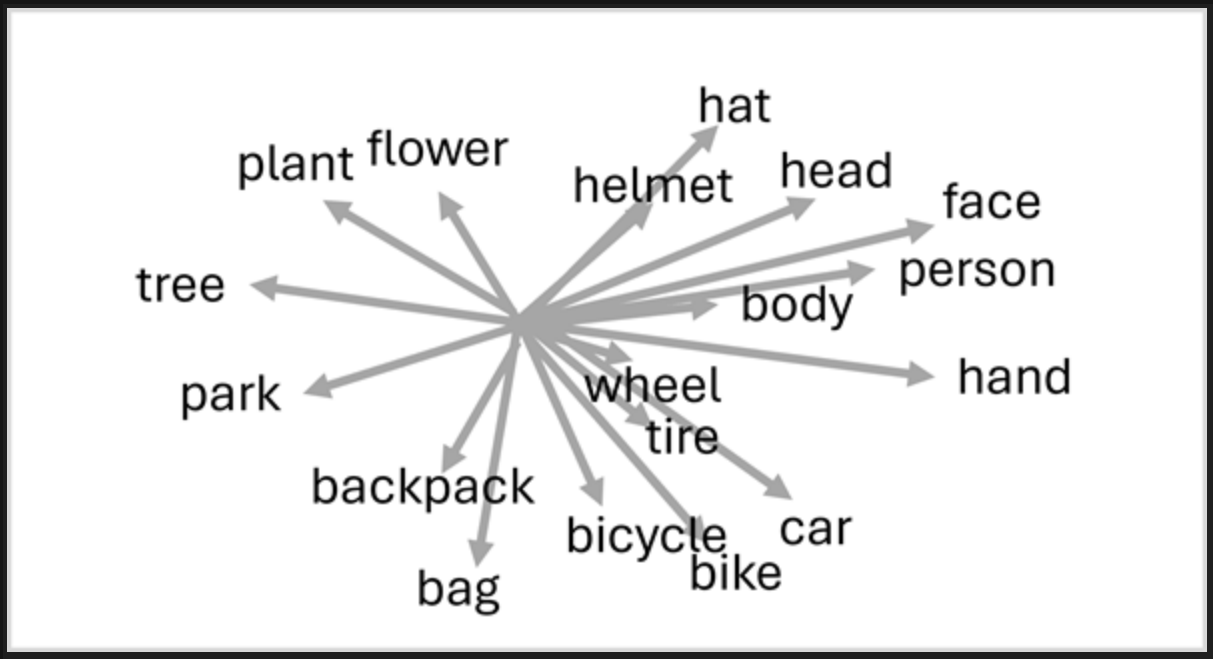
Vision transformers and multimodal models

CNNs have been at the core of computer vision solutions for many years. While they're commonly used to solve image classification problems as described previously, they're also the basis for more complex computer vision models. For example, *object detection* models combine CNN feature extraction layers with the identification of *regions of interest* in images to locate multiple classes of object in the same image. Many advances in computer vision over the decades have been driven by improvements in CNN-based models.

However, in another AI discipline - *natural language processing* (NLP), another type of neural network architecture, called a *transformer* has enabled the development of sophisticated models for language.

Semantic modeling for language - Transformers

Transformers work by processing huge volumes of data, and encoding language *tokens* (representing individual words or phrases) as vector-based *embeddings* (arrays of numeric values). A technique called *attention* is used to assign embedding values that reflect different aspects of how each token is used in the context of other tokens. You can think of the embeddings as vectors in multidimensional space, in which each dimension embeds a linguistic attribute of a token based on its context in the training text, creating semantic relationships between tokens. Tokens that are commonly used in similar contexts define vectors that are more closely aligned than unrelated words.



Tokens that are semantically similar are encoded in similar directions, creating a semantic language model that makes it possible to build sophisticated NLP solutions for text analysis, translation, language generation, and other tasks.

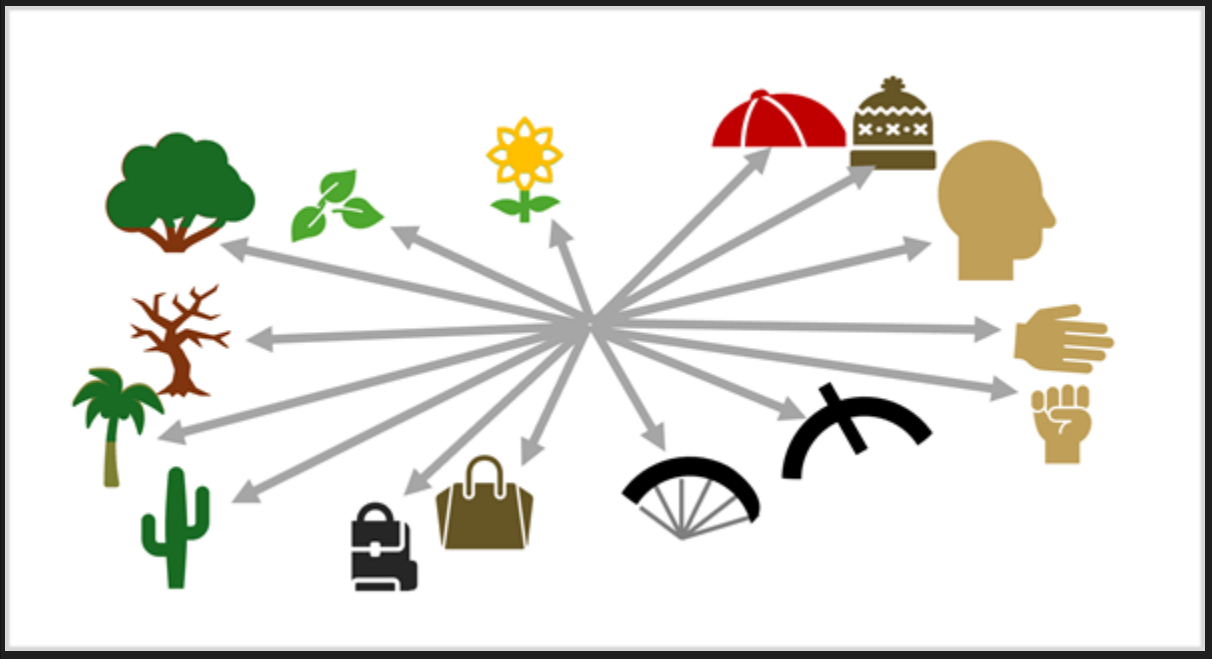
Note

In reality, encoders in transformer networks create vectors with many more dimensions, defining complex semantic relationships between tokens based on linear algebraic calculations. The math involved is complex, as is the architecture of a transformer model. Our goal here is just to provide a *conceptual* understanding of how encoding creates a model that encapsulates relationships between entities.

Semantic model for images - Vision transformers

The success of transformers as a way to build language models has led AI researchers to consider whether the same approach would be effective for image data. The result is the development of *vision transformer* (ViT) models, in which a model is trained using a large volume of images. Instead of encoding text-based tokens, the transformer extracts *patches* of pixel values from the image, and generates a linear vector from the pixel values.

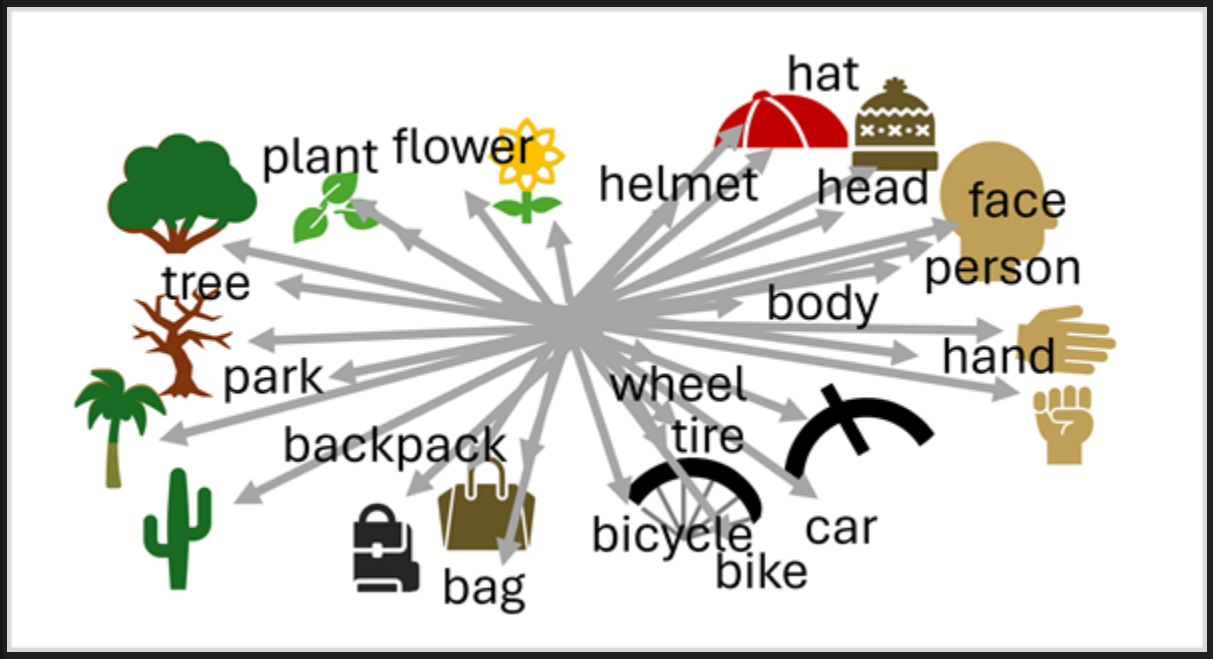
The same *attention* technique that's used in language models to embed contextual relationships between tokens, is used to determine contextual relationships between the patches. The key difference is that instead of encoding linguistic characteristics into the embedding vectors, the embedded values are based on visual features, like color, shape, contrast, texture, and so on. The result is a set of embedding vectors that creates a multidimensional "map" of visual features based on how they are commonly seen in the training images.



As with language models, the embeddings result in visual features that are used in similar context being assigned similar vector directions. For example, the visual features common in a *hat* may be contextually related to the visual features that are common in a *head*; because the two things are often seen together. The model has no understanding of what a "hat" or a "head" *is*; but it can infer a semantic relationship between the visual characteristics.

Bringing it all together - Multimodal models

A language transformer creates embeddings that define a linguistic vocabulary that encode semantic relationships between words. A vision transformer creates a visual vocabulary that does the same for visual features. When the training data includes images with associated text descriptions, we can combine the encoders from both of these transformers in a *multimodal* model; and use a technique called *cross-model attention* to define a unified spatial representation of the embeddings, like this.



This combination of language and vision embeddings enables the model to discern semantic relationships between language and visual features. This capability in turn enables the model to predict complex descriptions for images it hasn't previously seen, by recognizing visual features and searching the shared vector space for associated language.

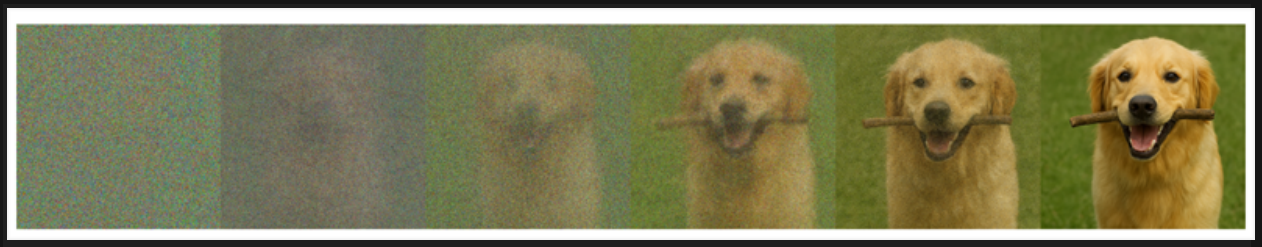
A person in a park with a hat and a backpack

Image generation

The same multimodal model architecture that enables AI to create natural language responses to visual input, can also be used to enable it to create images in response to natural language prompts. By identifying the visual features associated with language, an image synthesis model can take a description of a desired image or video and generate it.

Most modern image-generation models use a technique called *diffusion*, in which a prompt is used to identify a set of related visual features that can be combined to create an image. The image is then created iteratively, starting with a random set of pixel values and removing "noise" to create structure. After each iteration, the model evaluates the image so far to compare it to the prompt, until a final image that depicts the desired scene is produced.

For example, the prompt *"A dog carrying a stick in its mouth"* might result in a diffusion process with the following iterations:



Some models can apply a similar process to generating video. The video generation process uses the same technique to identify visual features that are associated with language tokens, but also takes into account factors like the physical behavior of objects in the real world (such as ensuring that a dog walks with its feet on the ground) and the temporal progression (so that the video depicts a logical sequence of activity).

Get started with computer vision in Azure

Introduction

Computer vision is a field of artificial intelligence (AI) that enables machines to interpret and understand visual information from the world—such as images, videos, and live camera feeds. Computer vision capabilities support the automation of time-intensive tasks and enable possibilities that did not exist before.

Consider some of these applications of computer vision:

- *Manufacturing – Defect Detection*: AI vision systems inspect products on assembly lines in real time. They detect surface defects, misalignments, or missing components using object detection and image segmentation, reducing waste and improving quality control.
- *Healthcare – Medical Imaging Analysis*: Computer vision helps radiologists analyze X-rays, MRIs, and CT scans. AI models can highlight anomalies like tumors or fractures, assist in early diagnosis, and reduce human error.
- *Retail – Shelf Monitoring*: Retailers use AI vision to monitor store shelves. Cameras detect when products are out of stock or misplaced, enabling real-time inventory updates and improving customer experience.
- *Transportation – Autonomous Vehicles*: Self-driving cars rely on computer vision to recognize road signs, lane markings, pedestrians, and other vehicles. This enables safe navigation and decision-making in dynamic environments.

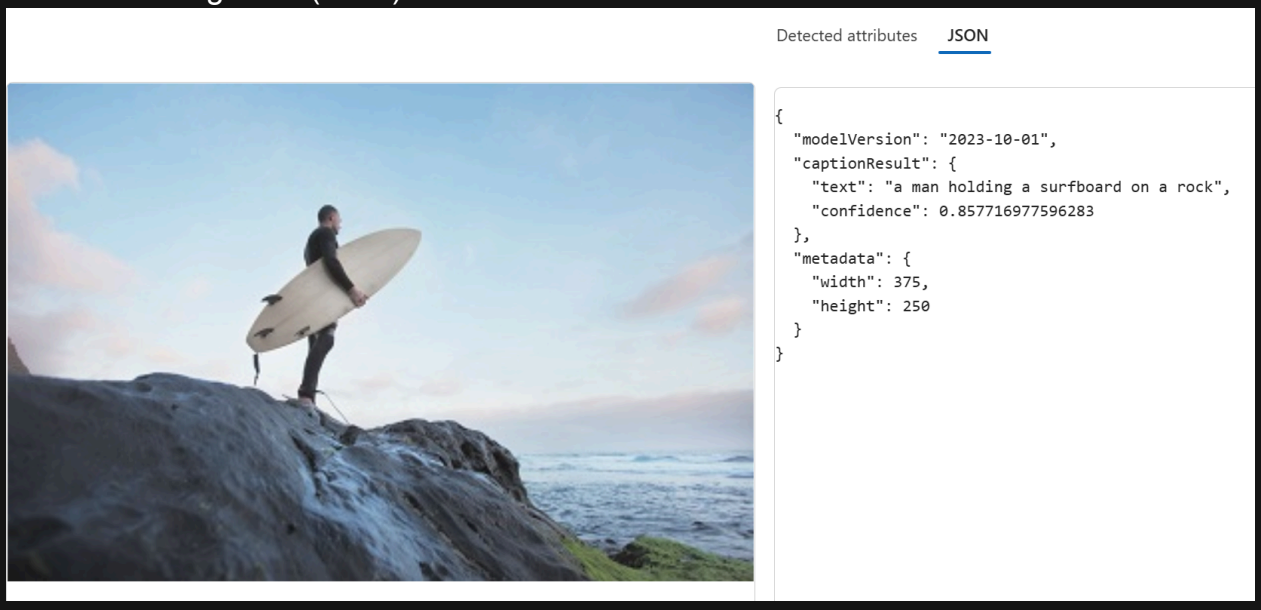
AI vision systems can be created using a range of Azure AI services. In this module we explore Microsoft Azure AI Vision, a cloud service that developers can use to create a wide range of computer vision solutions.

Understand Azure AI services for computer vision

Azure AI provides a wide range of cloud-based services for various AI tasks, including computer vision. Microsoft's Azure AI Vision service provides prebuilt and customizable computer vision models that are based on deep learning models and provide various capabilities. Azure AI Vision provides "off-the-shelf" functionality for many common computer vision scenarios, while retaining the ability to create custom models using your own images.

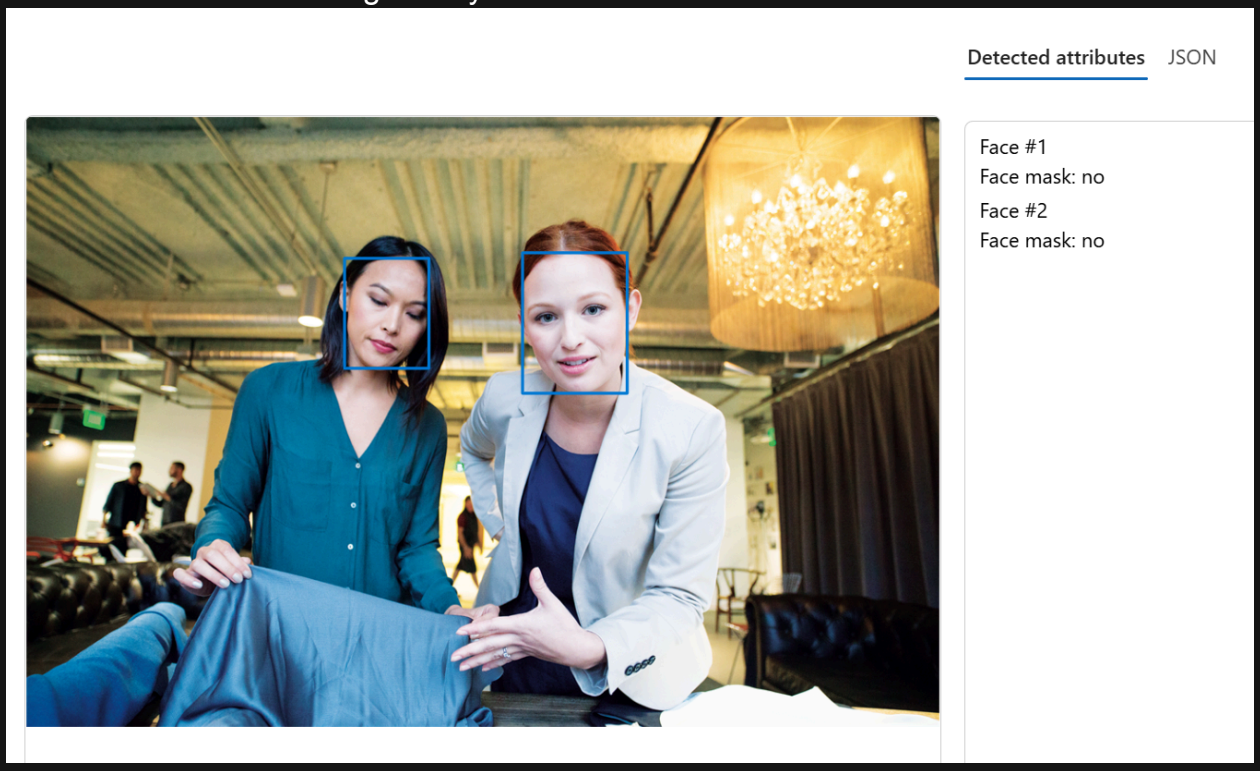
Azure AI Vision service contains several products. Within Azure AI Vision, there are services that handle specific sets of tasks including:

- Azure AI Vision Image Analysis service: Detects common objects in images, tags visual features, generates captions, and supports optical character recognition (OCR).



- Azure AI Face service: Detects, recognizes, and analyzes human faces in images. Provides specific models for facial analysis that extend beyond

what is available with image analysis.



There are many applications for Azure AI Vision's *image analysis* and *face* detection, analysis, and recognition. For example:

- Search engine optimization - using image tagging and captioning for essential improvements in search ranking.
- Content moderation - using image detection to help monitor the safety of images posted online.
- Security - facial recognition can be used in building security applications, and in operating systems for unlocking devices.
- Social media - facial recognition can be used to automatically tag known friends in photographs.
- Missing persons - using public cameras systems, facial recognition can be used to identify if a missing person is in the image frame.
- Identity validation - useful at ports of entry kiosks where a person holds a special entry permit.
- Museum archive management - using optical character recognition to preserve information from paper documents.

Note

Many modern vision solutions are built with a combination of capabilities. For example, video analysis capabilities are supported by *Azure AI Video indexer*. Azure AI Video indexer is built on several Azure AI services, such as Face, Translator, Image Analysis, and Speech.

Next, let's take a look at some core Azure AI Vision Image Analysis capabilities.

Understand Azure AI Vision Image Analysis capabilities

Azure AI Vision's image analysis capabilities can be used with or without customization. Some of the capabilities that do not require customization include:

- Describing an image with captions
- Detecting common objects in an image
- Tagging visual features
- Optical character recognition

Describing an image with captions

Azure AI Vision has the ability to analyze an image, evaluate the objects in it, and generate a human-readable description of the image. For example, consider the following image:

Azure AI Vision returns the following caption for this image:

A person jumping on a skateboard

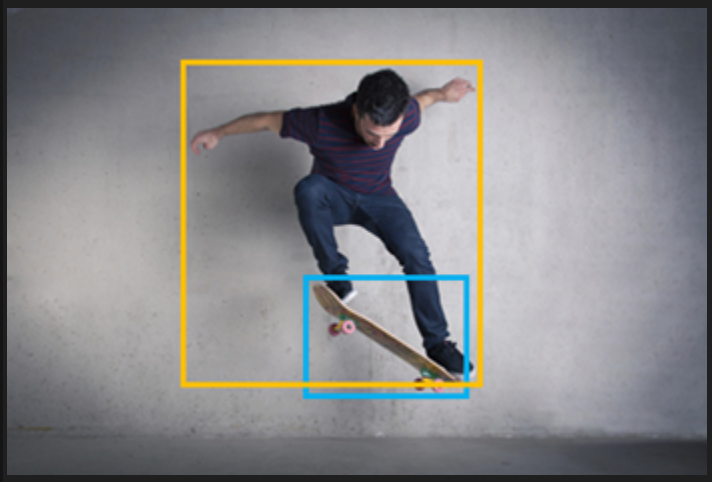
Detecting common objects in an image

Azure AI Vision can identify thousands of common objects in images. For example, when used to detect objects in the skateboarder image discussed previously, Azure AI Vision returns the following predictions:

- *Skateboard (90.40%)*
- *Person (95.5%)*

The predictions include a *confidence score* that indicates how confident the model is that what it describes is what is actually in the image.

In addition to the detected object labels and their probabilities, Azure AI Vision returns *bounding box* coordinates that indicate the top, left, width, and height of the object detected. You can use these coordinates to determine where in the image each object was detected, like this:



Tagging visual features

Azure AI Vision can suggest *tags* for an image based on its contents. Tags are associated with images as metadata. The tags summarize attributes of the image. You can use tags to index an image along with a set of key terms for a search solution.

For example, the tags returned for the skateboarder image (with associated confidence scores) include:

- *sport (99.60%)*
- *person (99.56%)*
- *footwear (98.05%)*
- *skating (96.27%)*
- *boardsport (95.58%)*
- *skateboarding equipment (94.43%)*
- *clothing (94.02%)*
- *wall (93.81%)*

- *skateboarding (93.78%)*
- *skateboarder (93.25%)*
- *individual sports (92.80%)*
- *street stunts (90.81%)*
- *balance (90.81%)*
- *jumping (89.87%)*
- *sports equipment (88.61%)*
- *extreme sport (88.35%)*
- *kickflip (88.18%)*
- *stunt (87.27%)*
- *skateboard (86.87%)*
- *stunt performer (85.83%)*
- *knee (85.30%)*
- *sports (85.24%)*
- *longboard (84.61%)*
- *longboarding (84.45%)*
- *riding (73.37%)*
- *skate (67.27%)*
- *air (64.83%)*
- *young (63.29%)*
- *outdoor (61.39%)*

Optical character recognition

Azure AI Vision service can use optical character recognition (OCR) capabilities to detect text in images. For example, consider the following image of a nutrition label on a product in a grocery store:

The Azure AI Vision service can analyze this image and extract the following text:

```
Nutrition Facts Amount Per Serving

Serving size:1 bar (40g)
```

```
Serving Per Package: 4

Total Fat 13g

Saturated Fat 1.5g

Amount Per Serving

Trans Fat 0g

calories 190

Cholesterol 0mg

ories from Fat 110

Sodium 20mg

ntDaily Values are based on

Vitamin A 50

calorie diet
```

Training custom models

If the built-in models provided by Azure AI Vision don't meet your needs, you can use the service to train a custom model for *image classification* or *object detection*. Azure AI Vision builds custom models on the pre-trained foundation model, meaning that you can train sophisticated models by using relatively few training images.

Image classification

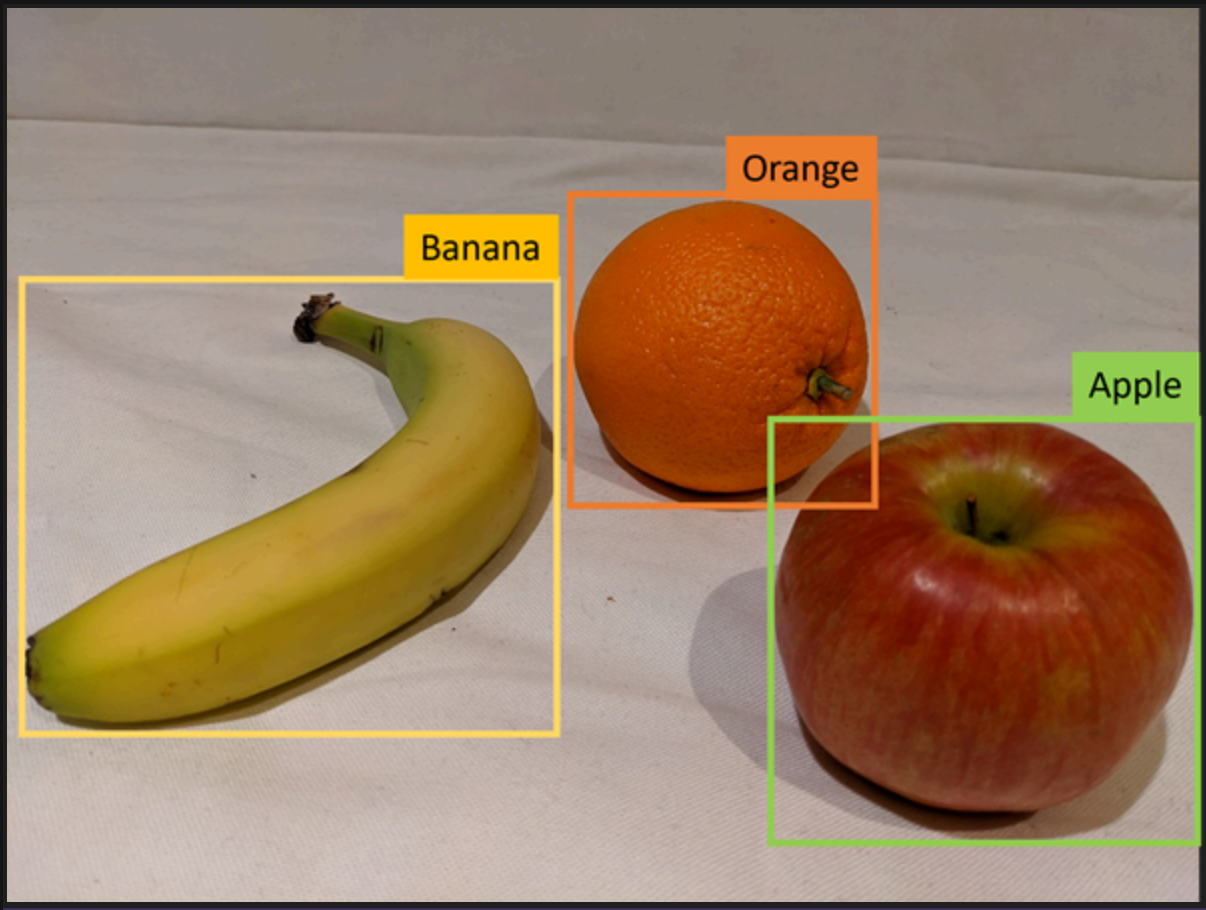
An image classification model is used to predict the category, or *class* of an image. For example, you could train a model to determine which type of fruit is shown in an image, like this:

Apple	Banana	Orange



Object detection

Object detection models detect and classify objects in an image, returning bounding box coordinates to locate each object. In addition to the built-in object detection capabilities in Azure AI Vision, you can train a custom object detection model with your own images. For example, you could use photographs of fruit to train a model that detects multiple fruits in an image, like this:



Note

Details of how to use Azure AI Vision to train a custom model are beyond the scope of this module. You can find information about custom model training in the Azure AI Vision documentation.

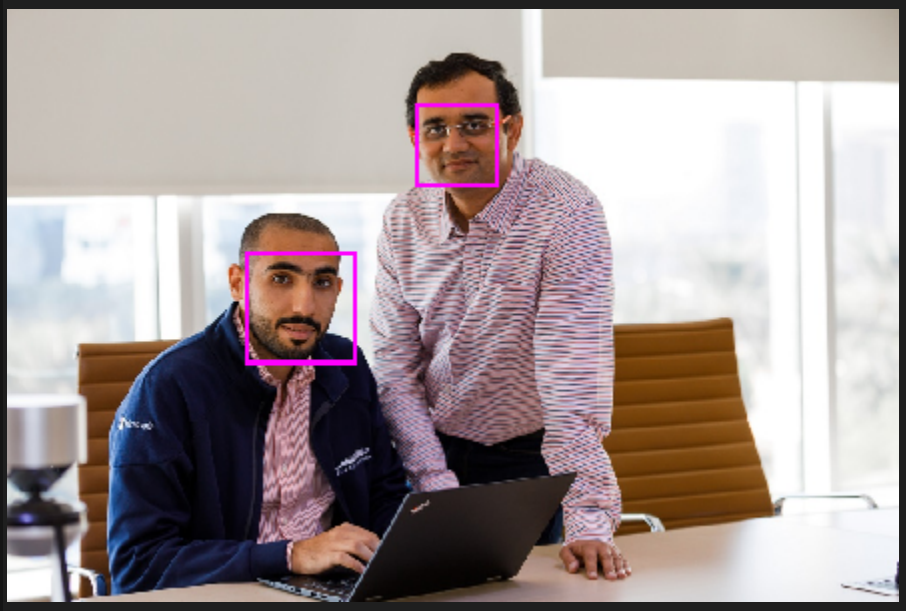
Next, let's look at capabilities specific to Azure AI Vision's Face service.

Understand Azure AI Vision's Face service capabilities

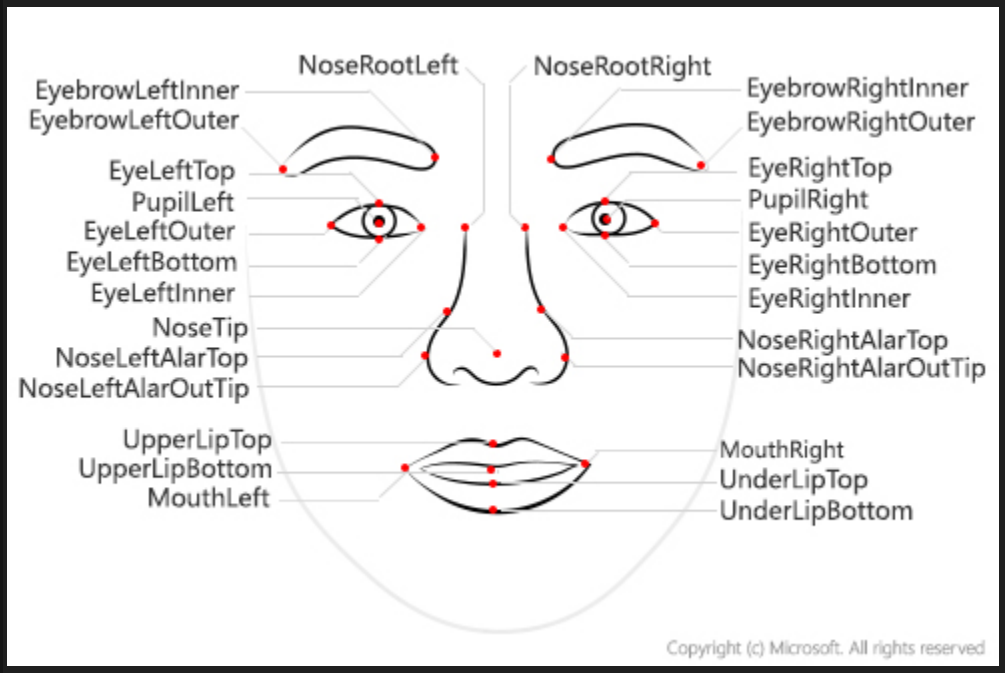
As a product within Azure AI Vision, Azure AI Face supports specific use cases such as verifying user identity, liveness detection, touchless access control, and face redaction. Several concepts, including face detection and recognition, are essential to working with Face.

Facial detection

Face detection involves identifying regions of an image that contain a human face, typically by returning *bounding box* coordinates that form a rectangle around the face, like this:



With Face, facial features can be used to train machine learning models to return other information, such as facial features such as nose, eyes, eyebrows, lips, and others.



Facial recognition

A further application of facial analysis is to train a machine learning model to identify known individuals from their facial features. This is known as *facial recognition*, and uses multiple images of an individual to train the model. This trains the model so that it can detect those individuals in new images on which it wasn't trained.

When used responsibly, facial recognition is an important and useful technology that can improve efficiency, security, and customer experiences.

Azure AI Face service capabilities

The Azure AI Face service can return the rectangle coordinates for any human faces that are found in an image, as well as a series of related attributes:

- Accessories: indicates whether the given face has accessories. This attribute returns possible accessories including headwear, glasses, and mask, with confidence score between zero and one for each accessory.
- Blur: how blurred the face is, which can be an indication of how likely the face is to be the main focus of the image.
- Exposure: such as whether the image is underexposed or over exposed. This applies to the face in the image and not the overall image exposure.
- Glasses: whether or not the person is wearing glasses.
- Head pose: the face's orientation in a 3D space.
- Mask: indicates whether the face is wearing a mask.
- Noise: refers to visual noise in the image. If you have taken a photo with a high ISO setting for darker settings, you would notice this noise in the image. The image looks grainy or full of tiny dots that make the image less clear.

- Occlusion: determines if there might be objects blocking the face in the image.
- Quality For Recognition: a rating of high, medium, or low that reflects if the image is of sufficient quality to attempt face recognition on.

Responsible AI use

Important

To support Microsoft's Responsible AI Standard, Azure AI Face and Azure AI Vision have a Limited Access policy.

Anyone can use the Face service to:

- Detect the location of faces in an image.
- Determine if a person is wearing glasses.
- Determine if there's occlusion, blur, noise, or over/under exposure for any of the faces.
- Return the head pose coordinates for each face in an image.

The Limited Access policy requires customers to submit an intake form to access additional Azure AI Face service capabilities including:

- Face verification: the ability to compare faces for similarity.
- Face identification: the ability to identify named individuals in an image.
- Liveness detection: the ability to detect and mitigate instances of recurring content and/or behaviors that indicate a violation of policies (e.g., such as if the input video stream is real or fake).

Next, let's take a look at how you can get started with Azure AI Vision.

Get started in Azure AI Foundry portal

Azure AI Vision provides the building blocks for incorporating vision capabilities into applications. As one of many Azure AI services, you can create solutions with Azure AI Vision in several ways including:

- The Azure AI Foundry portal
- A software development kit (SDK) or REST API

Azure resources for Azure AI Vision service

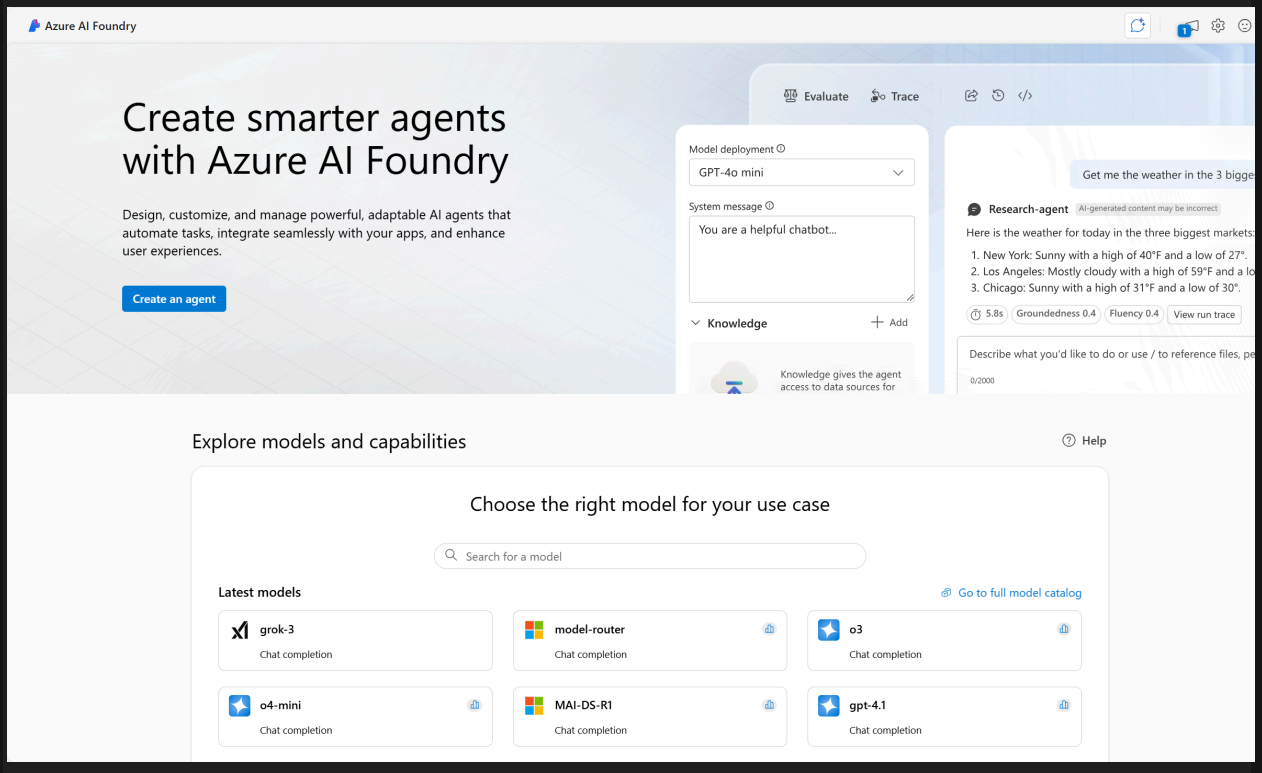
To use Azure AI Vision, you need to create a resource for it in your Azure subscription. You can use either of the following resource types:

- Azure AI Vision: A specific resource for the Azure AI Vision service. Use this resource type if you don't intend to use any other Azure AI services, or if you want to track utilization and costs for your Azure AI Vision resource separately.
- Azure AI services: A general resource that includes Azure AI Vision along with many other Azure AI services; such as Azure AI Language, Azure AI Custom Vision, Azure AI Translator, and others. Use this resource type if you plan to use multiple AI services and want to simplify administration and development.

Note

There are several ways to create resources with Azure. You can use a user interface to create resources or write a script. Both the Azure portal and Azure AI Foundry portal provide user interfaces for resource creation. Choose the Azure AI Foundry portal when you also want to see examples of Azure AI services in action.

Get started in Azure AI Foundry portal

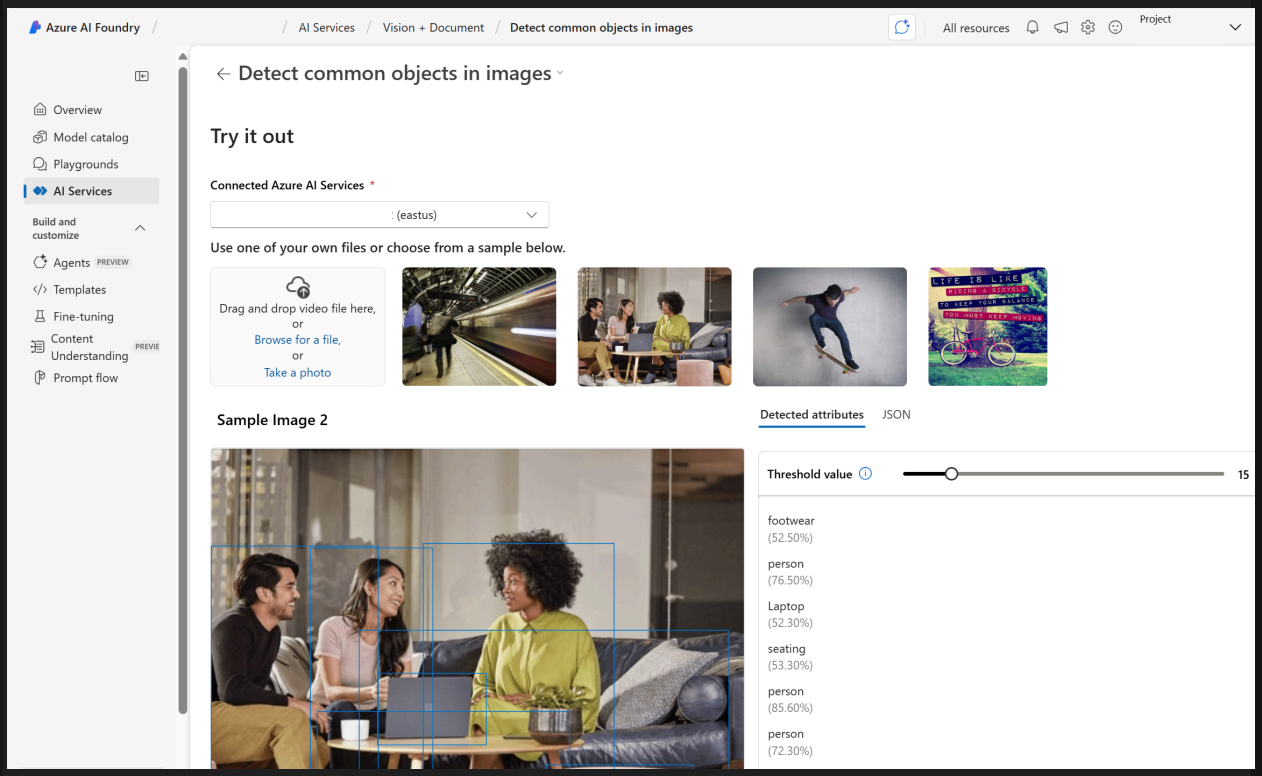


Azure AI Foundry provides a unified platform for enterprise AI operations, model builders, and application development. Azure AI Foundry portal provides a user

interface based around hubs and projects. To use any of the Azure AI services, including Azure AI Vision, you create a project in Azure AI Foundry, which will also create an Azure AI services resource for you.

Projects in Azure AI Foundry help you organize your work and resources effectively. Projects act as containers for datasets, models, and other resources, making it easier to manage and collaborate on AI solutions.

Within Azure AI Foundry portal, you have the ability to try out service features by testing with sample images or uploading your own.



Next let's try out Azure AI Vision in Azure AI Foundry portal.

Introduction to AI-powered information extraction concepts

Introduction

Today's organizations deal with all kinds of content such as documents, video, audio, images, and text. A common task in these organizations includes identifying and storing key information from the content into databases.

Consider some of these use cases:

- A manufacturer has images of each of its products. The images need to be analyzed for defects and anomalies.
- A business works with a high volume of invoices, contracts, and reports with charts. Key data and summaries from the documents need to be extracted and logged.
- Many hours of customer calls are recorded for quality purposes. The audio needs to be transcribed, summarized, and analyzed for sentiment.
- A streaming catalog contains a large volume of video. Important moments in each video need to be tagged with metadata based on their content.

Manually processing such content can be slow and potentially error-prone. AI-powered information extraction encompasses capabilities that extract meaning from content. In this module, you explore core concepts related to information extraction.

Overview

AI-powered information extraction and analysis enables organizations to gain actionable insights from data that might otherwise be locked up in documents, images, audio files, or other assets. Insights can come from structured and unstructured content. Structured content is information stored in a consistent format. Some examples include invoices, tax forms, and tables. Unstructured content is information that isn't in a predefined format. Some examples include emails, audio recordings, images, and videos.

Information extraction processes

In general, information extraction processes follow these steps:

Step	Description
Source Identification	Determine where the information resides and if it needs to be digitized.

Extraction	Leverages many techniques based on machine learning to understand and extract data from digitized content.
Transformation & Structuring	Extracted data is transformed into structured formats like JSON or tables.
Storage & Integration	The processed data is then stored in databases, data lakes, or analytics platforms for further use.

Both the type of content and type of insights needed from that content inform which techniques are necessary for information extraction. In this module we will take a look at the extraction of information with AI:

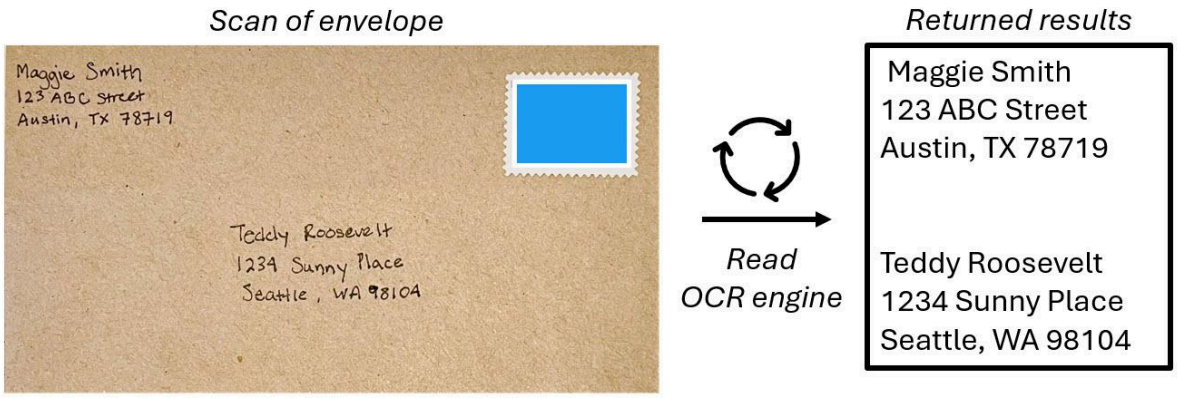
- From images
- From forms
- From multiple modalities
- For knowledge mining

In many ways, the techniques used for images, forms, multiple modalities, and knowledge mining build upon each other.

Understand the extraction of data from images

AI-powered information extraction replaces the need to manually inspect each piece of content for insights. Computer vision can extract insights from images to describe the people, places, things, and words they depict.

Computer vision is made possible by machine learning models that are trained to recognize features based on large volumes of existing images. Machine learning models process images by transforming the images into numerical information. At its core, vision models perform calculations on the numerical information, which result in predictions of what's in the images.



Optical Character Recognition (OCR) helps computers recognize that an element in an image contains text. OCR is the foundation of processing text in images and uses machine learning models that are trained to recognize individual shapes as letters, numerals, punctuation, or other elements of text. Much of the early work on implementing this kind of capability was performed by postal services to support automatic sorting of mail based on postal codes. Since then, the state-of-the-art for reading text has moved on, and we have models that detect printed or handwritten text in an image and digitize it line-by-line and word-by-word.

Note

The machine learning concepts associated with vision are covered in-depth in Introduction to computer vision concepts.

Next, let's see how data is extracted from forms with techniques that build upon OCR.

Understand the extraction of data from forms

Completed

100 XP

3 minutes

Forms and other documents have text data with *semantic meaning*. Semantic meaning refers to the intended meaning or interpretation of words, phrases, or symbols in a given context. Semantic meaning goes beyond just the literal definition of a word (syntax) and focuses on what the word or sentence actually conveys.

Document intelligence describes AI capabilities that process text and attach semantic meaning to the extracted text. As an extension of optical character recognition (OCR),

document intelligence automates the process of extracting and understanding information.

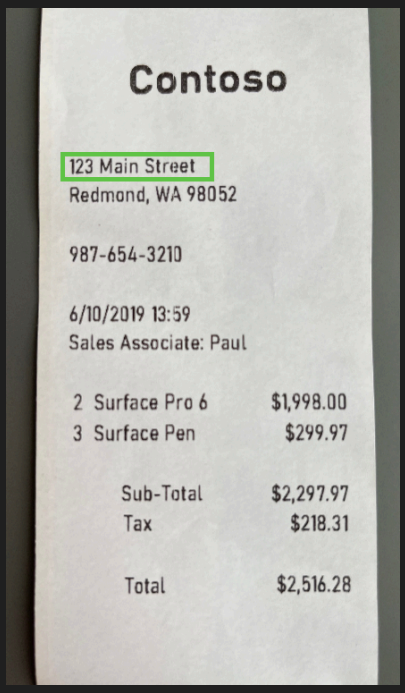
Consider an organization that needs to process large numbers of receipts for expenses claims, project costs, and other accounting purposes. Using document intelligence, the company can take a scanned image of a receipt, digitize the text with OCR, and extract semantic meaning. The semantic meaning of data in forms can be described in field-value pairs.

- The field name is the key or type of data entry.
- The field description is the definition of what the field name represents.
- The value corresponds with the field name and is the data specific to the content.

For example, in an invoice, the fields recognized may include:

- Name, address, and telephone number of the merchant
- Date and time of the purchase
- Name, quantity, and price of each item purchased
- Total, subtotals, and tax values

The data in forms is recognized with *bounding boxes*.



For example, the address information in on the receipt is saved as a `field name`, `address` and a `value`, `123 Main Street` with coordinates [4.1, 2.2], [4.3, 2.2], [4.3, 2.4], [4.1, 2.4]. Machine learning models can interpret the data in a document or form because they're trained to recognize patterns in bounding box coordinate locations.

The results of data extraction are associated with confidence levels for each field and data pair. This *confidence level* is a percentage between 0 and 1, indicating the likely level of accuracy. Data extracted with a high confidence score (closer to 1) could be relied on more confidently to actually represent what is in the original content.

Understand multimodal data extraction

AI-powered information extraction techniques can be combined to perform data extraction on multiple modalities of content, from documents to video and audio. Using multimodal data extraction can help with digital asset management, workflow automation, generating further insights, and more.

The orchestration of extraction techniques can include vision and document intelligence, and others including:

- Natural language processing can be used to find key phrases, entities, sentiment, etc. in written or spoken language.

> Note
>
> The machine learning concepts associated with NLP are covered in-depth in Introduction to natural language processing concepts.

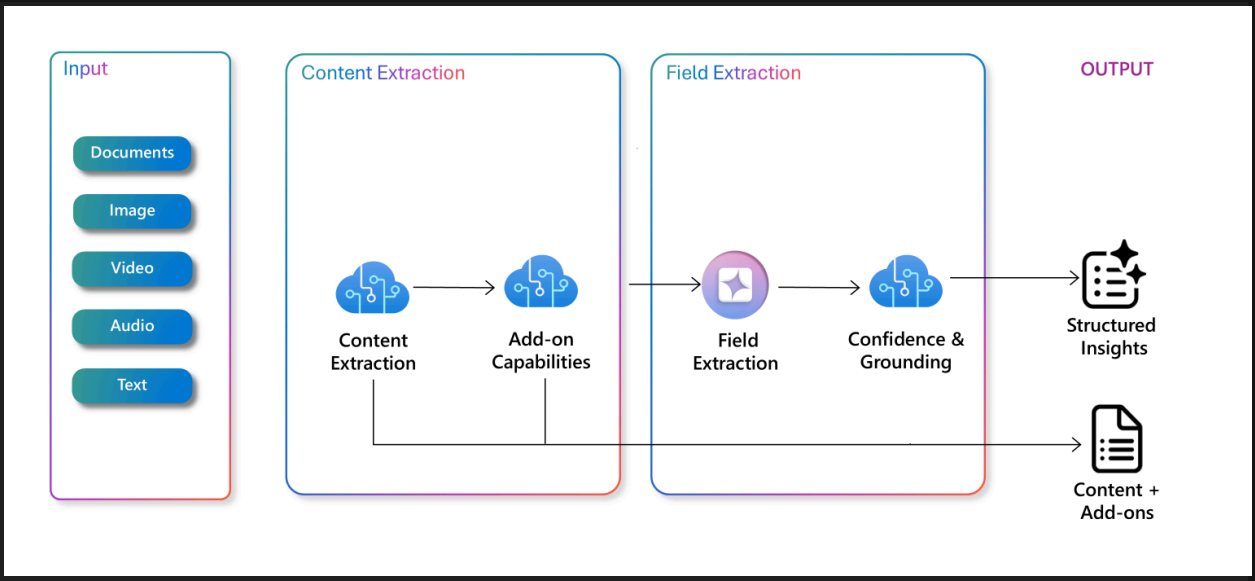
- Speech recognition takes the spoken word and converts it into data that can be processed - often by transcribing it into text. The spoken words can be in the form of a recorded voice in an audio file, or live audio from a microphone.

> Note
>
> Speech recognition is covered in Get started with speech on Azure.

- Generative AI can add to the data extraction process by allowing users to identify their own fields and field descriptions. It can be particularly useful when dealing with unstructured content. One example is the user-added *field* of "summary". The *value* associated with the field must be generated based on the data in the content.

> Note
>
> Generative AI concepts are covered in-depth in Introduction to generative AI on Azure.

The content processing pipeline for multimodal information extraction can include layers of these extraction techniques. One example of the pipeline's output is structured insights and additional generated content.



Understand data extraction for knowledge mining

Knowledge mining solutions provide automated information extraction from large volumes of often unstructured data. A foundational knowledge mining solution is search, the process of retrieving relevant information from a large dataset in response to a user query. AI-powered information extraction supports improvements in what is searchable in a search index.

In AI-powered information extraction for search, content first moves through Document cracking. Document cracking describes opening document formats like PDFs to extract the contents as ASCII text for analysis and indexing.

The contents then move through AI enrichment, which implements AI on your original content to extract more information. Examples of AI enrichment include adding captions to a photo and evaluating text sentiment. AI enriched content can be sent to a knowledge store, which persists output from an AI enrichment pipeline for independent analysis or downstream processing.

The resulting data is serialized as JSON data. The JSON populates the *search index*. The populated search index can be explored through queries. When users make a search query such as "coffee", the search engine looks for that information in the search index. A search index has a structure similar to a table, known as the index *schema*. A typical search index schema contains *fields*, the field's data type (such as string), and

field attributes. The fields store searchable text, and the field attributes allow for actions such as filtering and sorting. Below is an example of a search index schema:

```
{
  "name": "index",
  "fields": [
    {
      "name": "content", "type": "Edm.String", "analyzer": "standard.lucene", "fields": []
    }
    {
      "name": "keyphrases", "type": "Collection(Edm.String)", "analyzer": "standard.lucene", "fields": []
    },
    {
      "name": "imageTags", "type": "Collection(Edm.String)", "analyzer": "standard.lucene", "fields": []
    },
  ]
}
```

A result is a search solution which typically includes the following components:

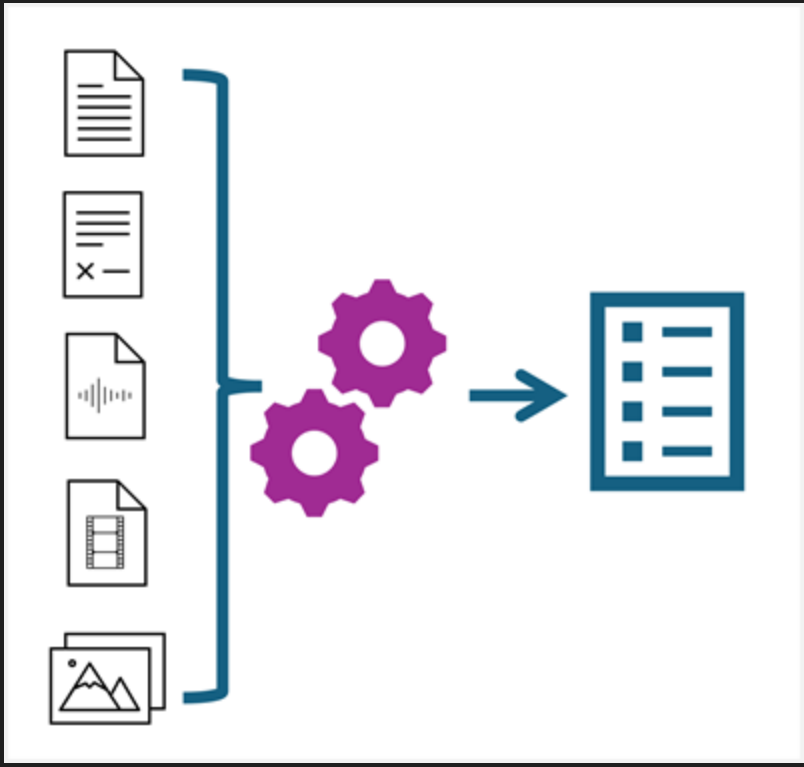
Component	Function
API Layer	Accepts user queries and routes them to the search engine.
Query Processor	Parses and interprets the query.
Search Strategies	Determines how to search—e.g., keyword, semantic, vector, or hybrid.
Execution Engine	Executes the query across the search index. AI-powered information extraction adds to the data that is searchable.
Result Aggregator	Combines results from multiple sources into a unified list.

Ranking Engine	Sorts results based on relevance, freshness, popularity, or AI signals.
Response Formatter	Formats the results for display in the user interface.

Get started with AI-powered information extraction in Azure

Introduction

AI-powered information extraction and analysis enables organizations to gain actionable insights from data that might otherwise be locked up in documents, images, audio files, or other assets.



Examples of information extraction scenarios include:

- A company needs to process employee expense claims, and has to extract expense descriptions and amounts from scanned receipts.
- A customer service agency wants to analyze recorded support calls to identify common problems and resolutions.
- A historical society needs to extract and store data from census records in scanned historical documents.
- A tourist organization wants to analyze video footage and images taken at popular sites to help estimate visitor volumes and improve capacity planning for tours.
- A finance department in a large corporation wants to automate accounts-payable processing by routing invoices received centrally to the appropriate departments for payment.
- A marketing organization wants to analyze a large volume of digital images and documents, extracting and indexing the extracted data so it can be easily searched.

Azure AI includes multiple services that can be used, individually or in combination, so support these kinds of information extraction scenarios. In this module, we'll explore some of these services and their capabilities.

Azure AI services for information extraction

Azure AI provides a wide range of cloud-based services for various AI tasks, including the extraction and analysis of information from digital content.

Core services used in information extraction scenarios include:

Service	Description
 Azure AI Vision Image Analysis	Azure AI Vision Image Analysis enables you to extract insights from images, including the detection and identification of common objects in images, the generation of relevant captions and tags for images, and the extraction of text in images.

Azure AI Content Understanding

Azure AI Content Understanding is a generative AI-based multimodal analysis service that can extract insights from structured documents, images, audio, and video.

Azure AI Document Intelligence

Azure AI Document Intelligence is designed to extract fields and values from digital (or digitized) forms, such as invoices, receipts, purchase orders, and others.

Azure AI Search

Azure AI Search performs AI-assisted indexing in which a pipeline of AI *skills* are used to systematically extract and index information from structured and unstructured content.

You can use each of these services separately, or combine them to build comprehensive solutions for:

- Data capture: Intelligently scanning images to capture and store data values. For example, using a cellphone camera to extract contact information from a business card.
- Business process automation: Reading data from forms and using it to trigger workflows. For example, extracting cost center and billing information from invoices and routing them to the appropriate accounts-payable department for processing.

- Meeting summarization and analysis: Analyzing and summarizing key points from recorded phone conversations or video conference calls. For example, automating note-taking and action assignments for a team meeting.
- Digital asset management (DAM): Managing digital assets like images or videos by automatically tagging and indexing them. For example, to create a searchable library of stock photographs.
- Knowledge Mining: Extracting key information from structured and unstructured data to be used for further analysis and reporting. For example, compiling census data from scanned records to populate a database.

Extract information with Azure AI Vision

The Azure AI Vision Image Analysis service is a great choice when you need to extract insights from photographs or small scanned documents, such as business cards or menus.

Automated caption and tag generation

You can use Azure AI Vision Image Analysis to generate descriptive text associated with an image. The service can analyze an image and generate:

- A caption that describes the image.
- A set of suggested dense captions for the key objects in the image.
- A collection of tags that help categorize the image.

For example, suppose you want to capture the key details related to this image:

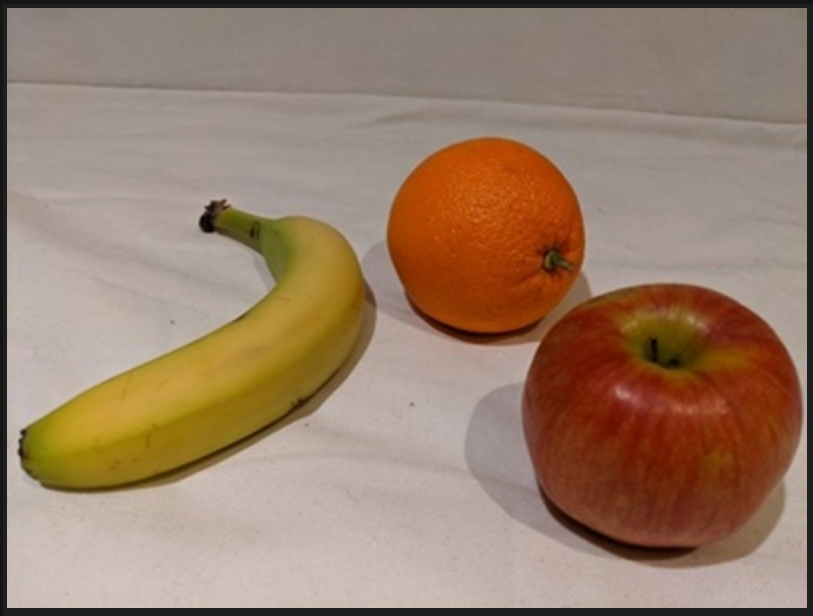
The AI Vision Image Analysis service generates the following descriptive text values.

- Caption: A man walking a dog on a leash
- Dense captions:
    - A man walking a dog on a leash
    - A man walking on the street
    - A yellow car on the street
    - A yellow car on the street
    - A green telephone booth with a green sign
- Tags:
    - outdoor
    - land vehicle
    - vehicle
    - building
    - road
    - street
    - wheel
    - taxi
    - person
    - clothing
    - car
    - dog
    - yellow
    - walking
    - city

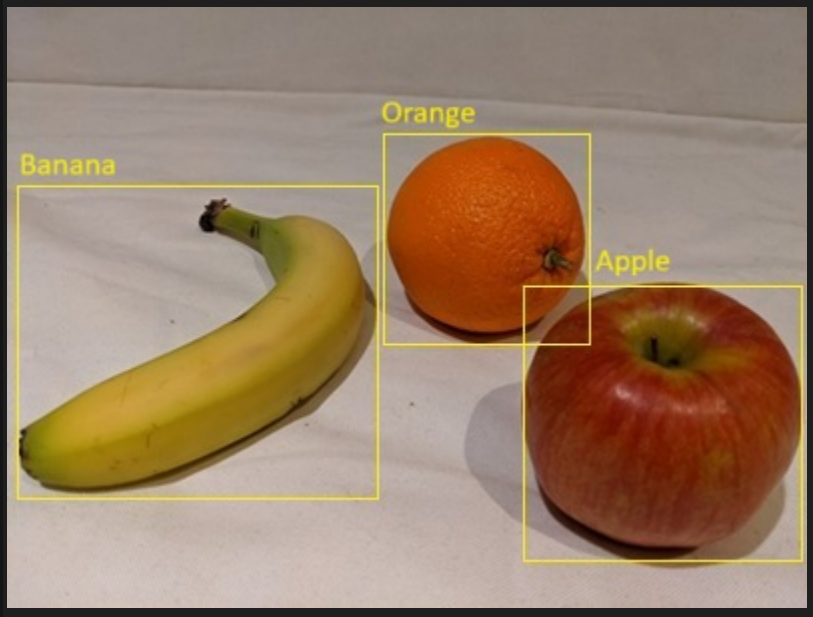
Object detection

Azure AI Vision Image Analysis can also detect common objects and people in an image.

For example, consider the following image:

 .

Azure AI Vision Image Analysis detects the types and locations of objects in this image, as shown here:



.

Optical character recognition (OCR)

When an image contains printed or handwritten text, Azure AI Vision Image Analysis can use a technique called *optical character recognition* (OCR) to determine the location and contents of each *line* of text, and each individual *word*. The OCR capabilities of Azure AI Vision Image Analysis are useful when you need to read text in an image for further processing, for example to translate a menu using a cellphone

application. Azure AI Vision Image Analysis can also be useful to extract small volumes of free-form text from simple documents; for example, to extract contact details from a business card.

Consider the following scanned business card:

You could use Azure AI Vision Image Analysis to locate and extract the text from this card, with the following results:

text

```
Adventure Works Cycles

Roberto Tamburello

Engineering Manager

roberto@adventure-works.com

555-123-4567
```

Extract multimodal information with Azure AI Content Understanding

Azure AI Content Understanding uses state-of-the-art AI models to analyze content in multiple formats, including:

- Text-based forms and documents
- Audio
- Images
- Video

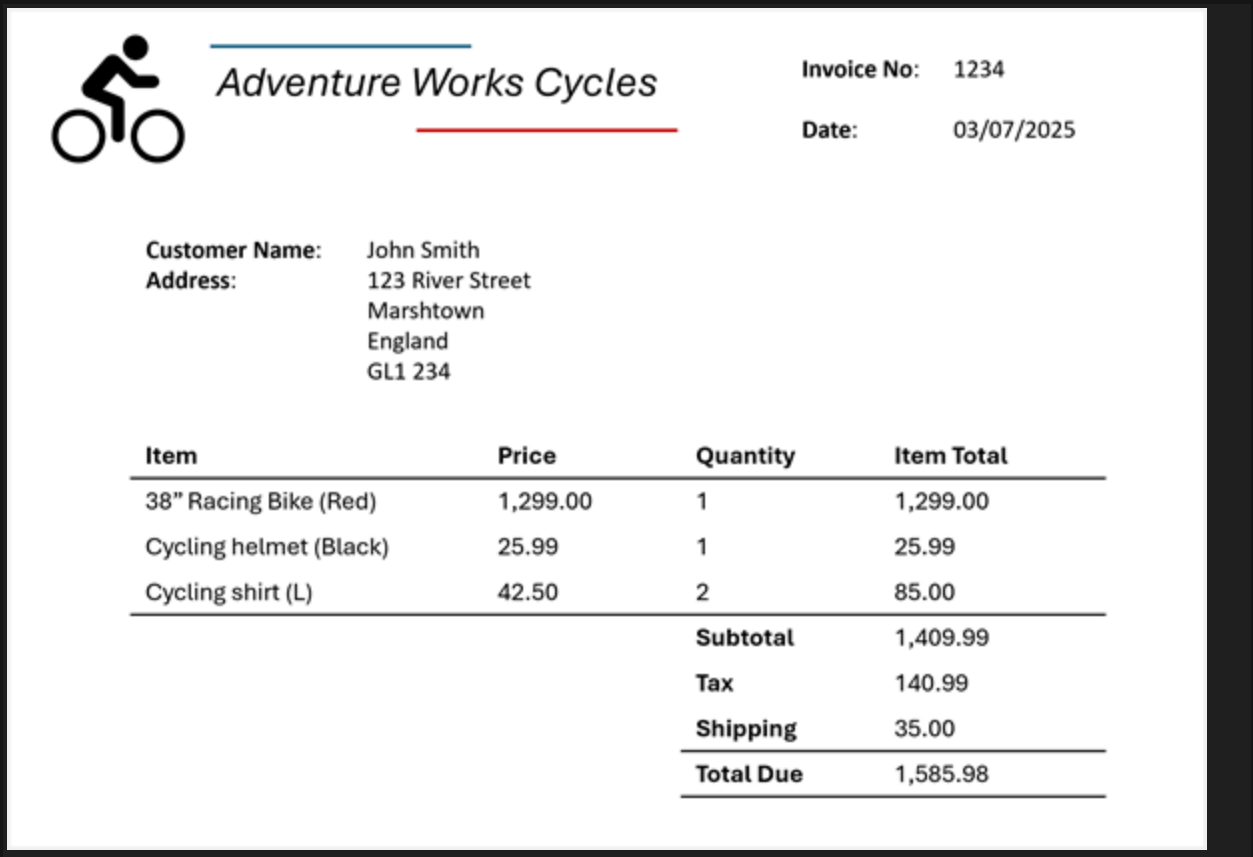
Analyzing forms and documents

Azure AI Content Understanding's document analysis capabilities go beyond simple OCR-based text extraction to include schema-based extraction of fields and their values.

For example, suppose you define a schema that includes the common fields typically found in an invoice, such as:

- Vendor name
- Invoice number
- Invoice date
- Customer name
- Custom address
- Items - the items ordered, each of which includes:
    - Item description
    - Unit price
    - Quantity ordered
    - Line item total
- Invoice subtotal
- Tax
- Shipping Charge
- Invoice total

Now suppose you need to extract this information from the following invoice:



Adventure Works Cycles

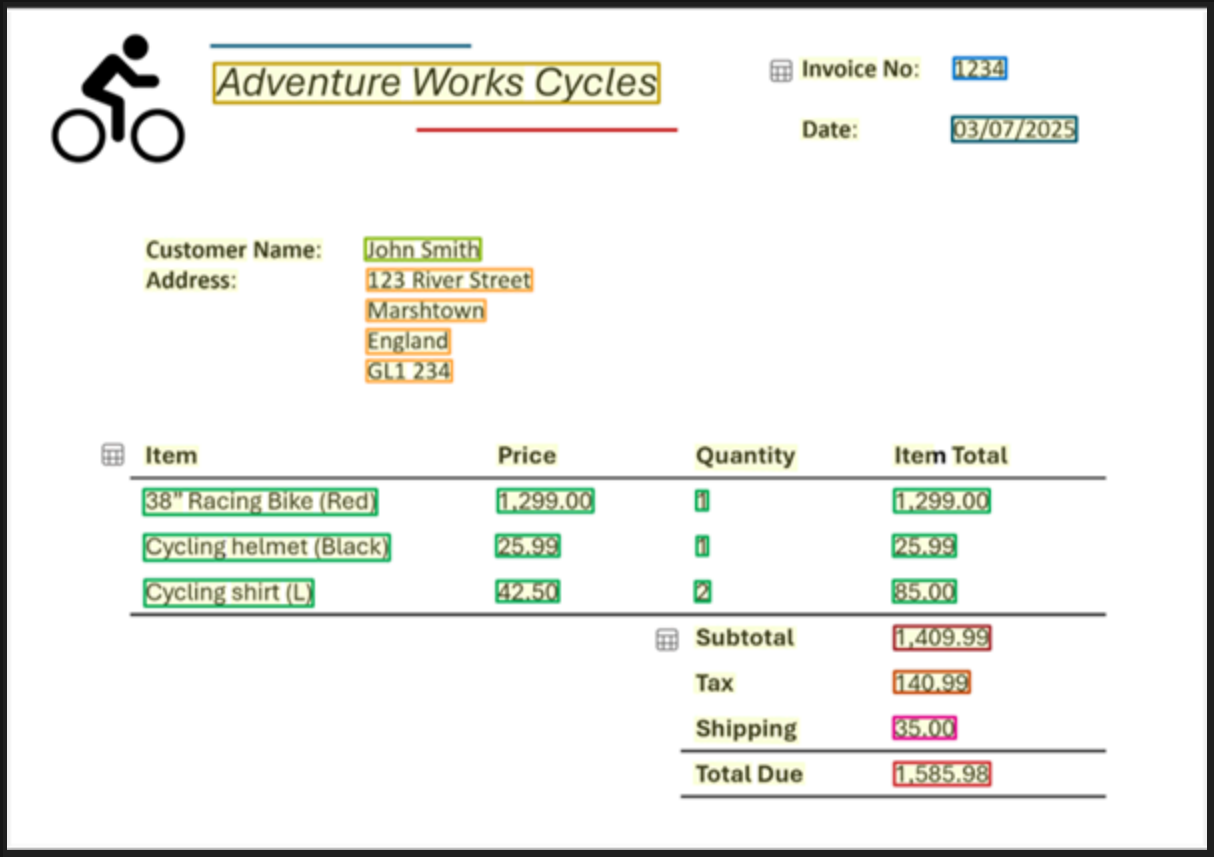
Invoice No: 1234

Date: 03/07/2025

Customer Name: John Smith
Address: 123 River Street
Marshtown
England
GL1 234

Item	Price	Quantity	Item Total
38" Racing Bike (Red)	1,299.00	1	1,299.00
Cycling helmet (Black)	25.99	1	25.99
Cycling shirt (L)	42.50	2	85.00
		Subtotal	1,409.99
		Tax	140.99
		Shipping	35.00
		Total Due	1,585.98

Azure AI Content Understanding can apply the invoice schema to your invoice and identify the corresponding fields, even when they're labeled with different names (or not labeled at all). The resulting analysis produces a result like this:



For each detected field, the value is extracted from the invoice:

- Vendor name: Adventure Works Cycles
- Invoice number: 1234
- Invoice date: 03/07/2025
- Customer name: John Smith
- Custom address: 123 River Street, Marshtown, England, GL1 234
- Items:
    - Item 1:
        - Item description: 38" Racing Bike (Red)
        - Unit price: 1299.00
        - Quantity ordered: 1
        - Line item total: 1299.00
    - Item 2:
        - Item description: Cycling helmet (Black)
        - Unit price: 25.99
        - Quantity ordered: 1
        - Line item total: 25.99
    - Item 3:
        - Item description: Cycling shirt (L)
        - Unit price: 42.50
        - Quantity ordered: 2

        - Line item total: 85.00
- Invoice subtotal: 1409.99
- Tax: 140.99
- Shipping Charge: 35.00
- Invoice total: 1585.98

Analyzing audio

In addition to text-based documents, Azure AI Content Understanding is capable of analyzing audio files to provide transcriptions, summaries, and other key insights.

Suppose you want to have AI summarize your voice mail. You might define a schema of key insights to extract from each recorded call, like this:

- Caller
- Message summary
- Requested actions
- Callback number
- Alternative contact details

Now suppose, a caller leaves you the following voice message:

```
Hi, this is Ava from Contoso.


Just calling to follow up on our meeting last week.


I wanted to let you know that I've run the numbers and I think we can
meet your price expectations.


Please call me back on 555-12345 or send me an e-mail at
Ava@contoso.com and we'll discuss next steps.


Thanks, bye!
```

Using Azure AI Content Understanding to analyze the audio recording and apply your schema produces the following results:

- Caller: Ava from Contoso

- Message summary: Ava from Contoso called to follow up on a meeting and mentioned that they can meet the price expectations. She requested a callback or an email to discuss next steps.
- Requested actions: Call back or send an email to discuss next steps.
- Callback number: 555-12345
- Alternative contact details: Ava@contoso.com

Analyzing images and video

Azure AI Content Understanding supports analysis of images and video to extract information based on a custom schema. For example, you could analyze images of a video conference to extract details of attendance, location, and other information.

Suppose you defined the following schema for an image taken by a collaborative messaging system that combines in-room attendees and remote attendees on a conference call system:

- Location
- In-person attendees
- Remote attendees
- Total attendees

You can use Azure AI Content Understanding to analyze the following still image from the conference room camera:

When applying the preceding schema to this image, Azure AI Content Understanding produces the following results:

- Location: Conference room
- In-person attendees: 1
- Remote attendees: 3
- Total attendees: 4

If instead of analyzing the still image, you were to create an analyzer for recorded video of the call; the schema could include attendance counts at various time intervals, details of who spoke during the call and what they said, a summary of the discussion, and a list of assigned actions from the meeting.

Extract information from forms with Azure AI Document Intelligence

Azure AI Document Intelligence is designed to support complex document and form processing scenarios. While you can also use Azure AI Content Understanding to extract fields from forms and documents, Azure AI Document Intelligence offers a large library of prebuilt models, from simple receipts to complex tax forms. You can also create sophisticated custom models of your own.

Using prebuilt models

Let's explore an example of using Azure AI Document Intelligence to extract data from a form.

Suppose a financial loan company needs to process hundreds of mortgage applications each day. Here's an example of just the first page of a standard 11-page mortgage application form:

To be completed by the ***Lender:***
Lender Loan No./Universal Loan Identifier 265fs6fAZ2H4984Gs4f23hQ811 Agency Case No. 115894

Uniform Residential Loan Application

Verify and complete the information on this application. If you are applying for this loan with others, each additional Borrower must provide information as directed by your Lender.

Section 1: Borrower Information. This section asks about your personal information and your income from employment and other sources, such as retirement, that you want considered to qualify for this loan.

1a. Personal Information

Name *(First, Middle, Last, Suffix)*
Gwen Stacy

Alternate Names – *List any names by which you are known or any names under which credit was previously received (First, Middle, Last, Suffix)*

Social Security Number 557 - 99 - 7283
(or Individual Taxpayer Identification Number)

Date of Birth *(mm/dd/yyyy)*
04 / 07 / 1989

Citizenship
- (●) U.S. Citizen
- ( ) Permanent Resident Alien
- ( ) Non-Permanent Resident Alien

Type of Credit
- ( ) I am applying for **individual credit.**
- (●) I am applying for **joint credit.** Total Number of Borrowers: 2
  Each Borrower intends to apply for joint credit. **Your initials:** ____

List Name(s) of Other Borrower(s) Applying for this Loan
(First, Middle, Last, Suffix) – Use a separator between names
Carmen Leannon

Marital Status
- (●) Married
- ( ) Separated
- ( ) Unmarried
  (Single, Divorced, Widowed, Civil Union, Domestic Partnership, Registered Reciprocal Beneficiary Relationship)

Dependents *(not listed by another Borrower)*
Number 2
Ages 10, 11

Contact Information
Home Phone (489) 746 – 8900
Cell Phone (831) 728 – 4766
Work Phone ( ) – **Ext.**
Email gwen19890407@outlook.com

Current Address
Street 1634 W Glenoaks Blvd Unit #
City Glendale State CA ZIP 91201 Country United States
How Long at Current Address? 1 Years 2 Months **Housing** ( ) No primary housing expense (●) Own ( ) Rent ($ /month)

If at Current Address for LESS than 2 years, list Former Address ☐ ***Does not apply***
Street 3533 Rosa View Unit # 5
City Richmondport State CA ZIP 94804 Country United States
How Long at Former Address? 5 Years 6 Months **Housing** ( ) No primary housing expense ( ) Own (●) Rent ($ 1,600.00 /month)

Mailing Address – *if different from Current Address* ☑ ***Does not apply***
Street Unit #
City State ZIP Country

1b. Current Employment/Self-Employment and Income ☐ *Does not apply*

Employer or Business Name Spider Web Corp. Phone (390) 353 – 2474
Street 3533 Bandini Ave Unit #
City Glendale State CA ZIP 92506 Country United States

Position or Title Language Teacher
Start Date 01 / 08 / 2020 *(mm/dd/yyyy)*
How long in this line of work? 1 Years 2 Months

Check if this statement applies:
☐ I am employed by a family member, property seller, real estate agent, or other party to the transaction.

☐ **Check if you are the Business Owner or Self-Employed**
( ) I have an ownership share of less than 25%.
( ) I have an ownership share of 25% or more.
Monthly Income (or Loss) $

Gross Monthly Income

Base	$ 4,204.00	/month
Overtime	$	/month
Bonus	$	/month
Commission	$	/month
Military Entitlements	$	/month
Other	$ 50.00	/month
TOTAL $	**4,254.00**	**/month**

Azure AI Document Intelligence includes a prebuilt model for this type of form, making it easy to build a solution that can locate and extract fields, such as:

- Borrower Name
- Address
- Telephone number
- Social security number
- Date of birth
- Marital status
- Employment status
- Employer name
- Employer address
- Income
- Citizenship
- *and more*

Creating custom models

With Azure AI Document Intelligence, you can train custom models by using labeled examples of the documents you want to analyze. Labeling your documents involves using OCR to define the *layout* of your document and identifying the discrete *fields* in your documents that you want to extract.

Create a knowledge mining solution with Azure AI Search

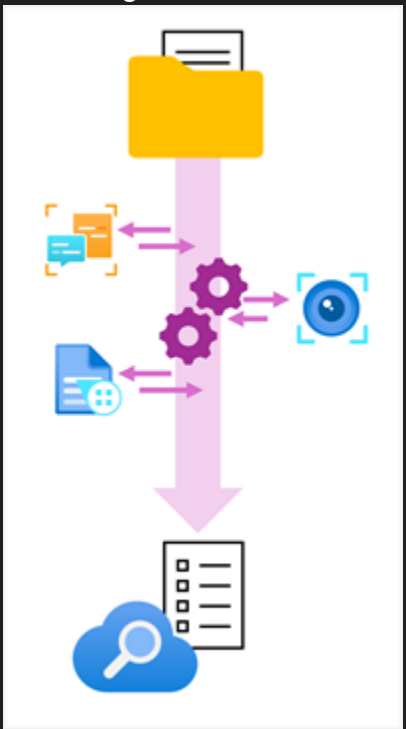
Fundamentally, Azure AI Search is a cloud service for indexing and searching data. However, its use of AI *skills* to extract insights from multiple formats of data and the ability to integrate it with other AI services, including Azure AI Vision and Azure AI Document Intelligence make it a powerful platform for building digital asset management and knowledge mining solutions.

Indexers, indexes, and skills

At the heart of an Azure AI Search solution is an *indexer*, which defines a repeatable process to:

1. Ingest data from a *source*, such as an Azure Storage container of documents or a database.

2. *Crack* documents to extract their contents - for example, retrieving the text and image data in a PDF document.
3. Apply a sequence of tasks to retrieve information from the data and generate a hierarchy of *fields* for the index. Some fields are core attributes of the source data (for example document file names and last saved dates), while others are generated by using AI *skills*. For example:
    - Using Azure AI Vision services to generate *tags* and *captions* for images.
    - Using Azure AI Language services to derive fields for *sentiment* or *named entities*.
    - Using Azure AI Document Intelligence to extract field values from forms.
4. Persisting the extracted fields as an *index*.



The resulting index can be used to enable users to search for information in the extracted fields based on keywords and filtering criteria.

Persisting extracted data to a knowledge store

As well as creating a searchable index, Azure AI Search can persist the extracted data assets to a *knowledge store* in Azure Storage.

The indexer can save the following kinds of asset in a knowledge store:

- Tables of field values.

- Images extracted from documents.
- JSON documents representing data structures; which can be complex hierarchies of fields and values.