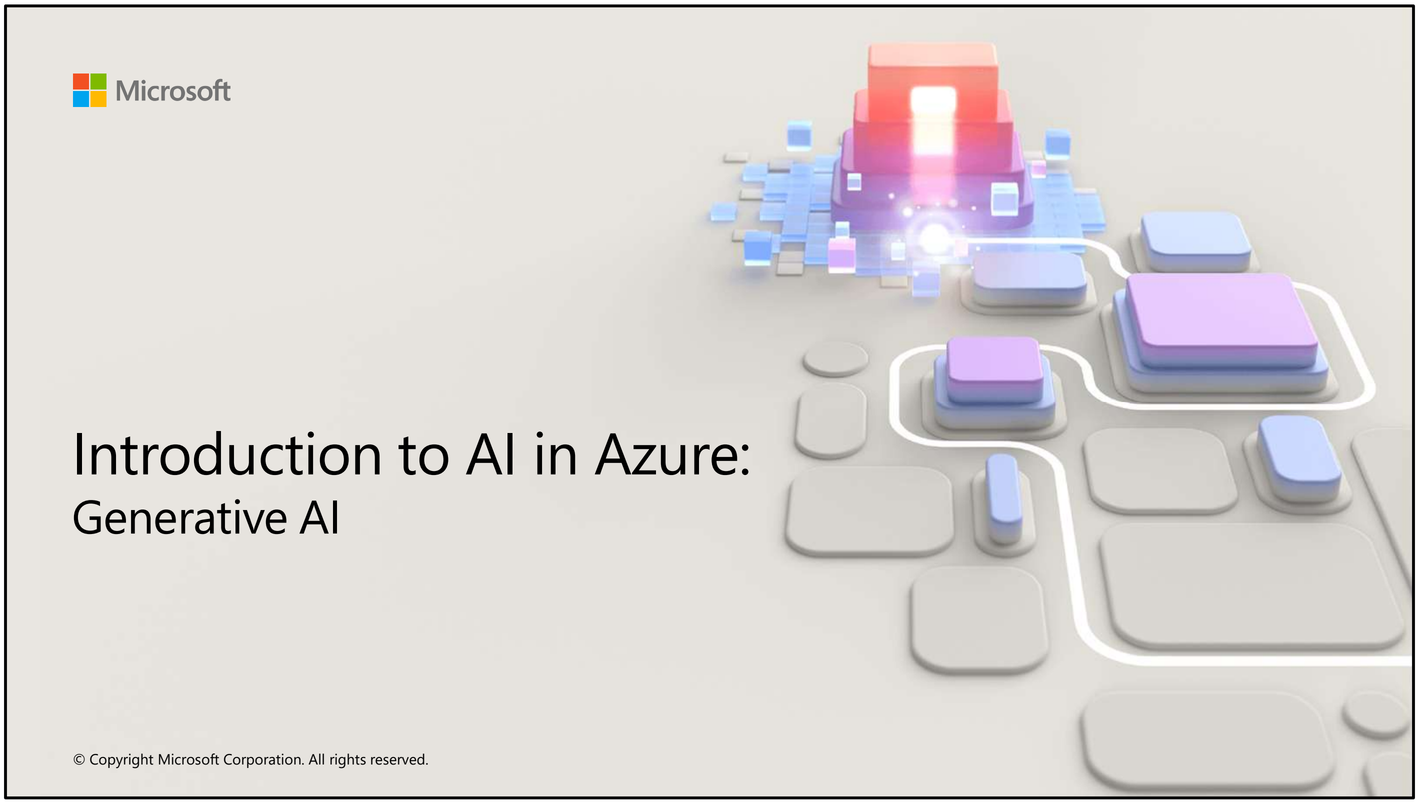


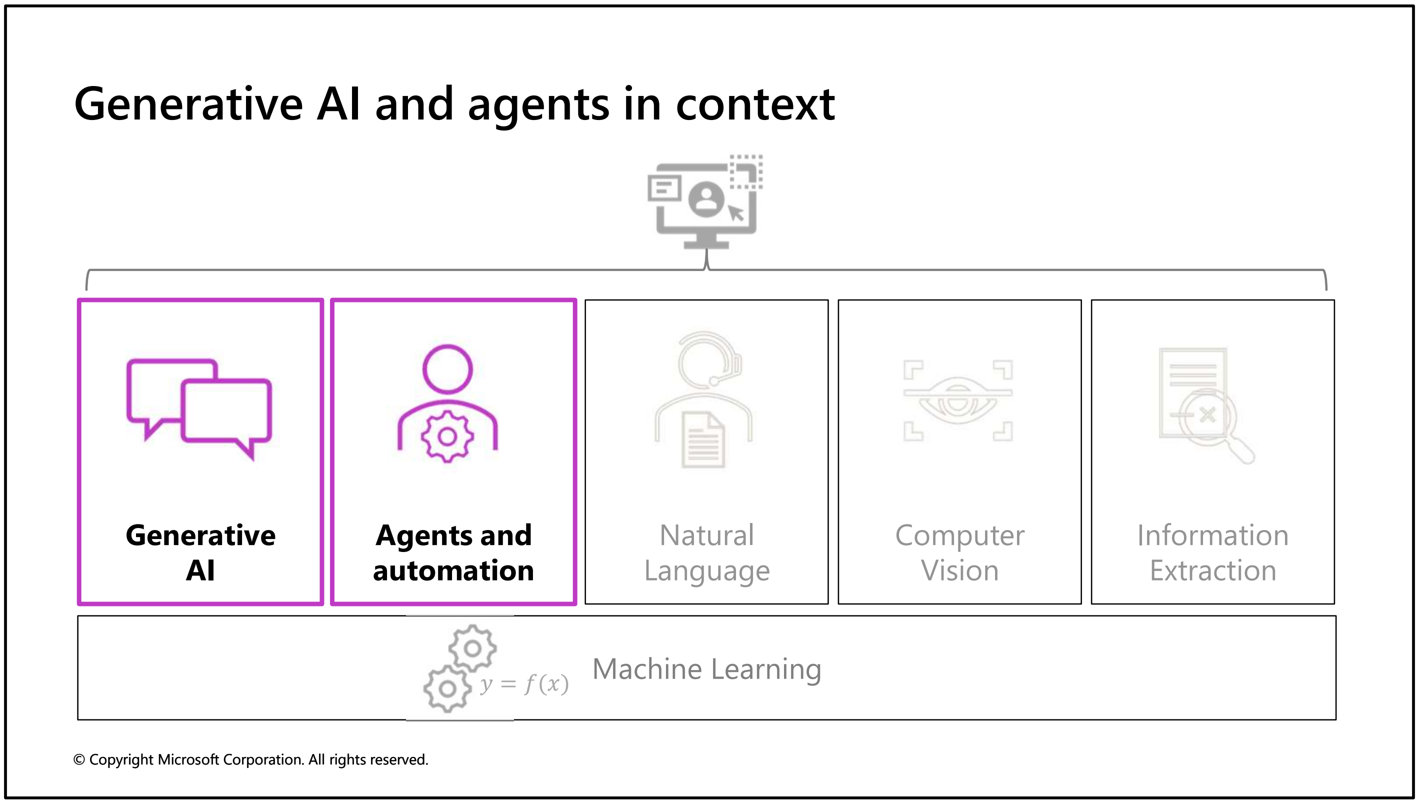
Microsoft

Introduction to AI in Azure:

Generative AI

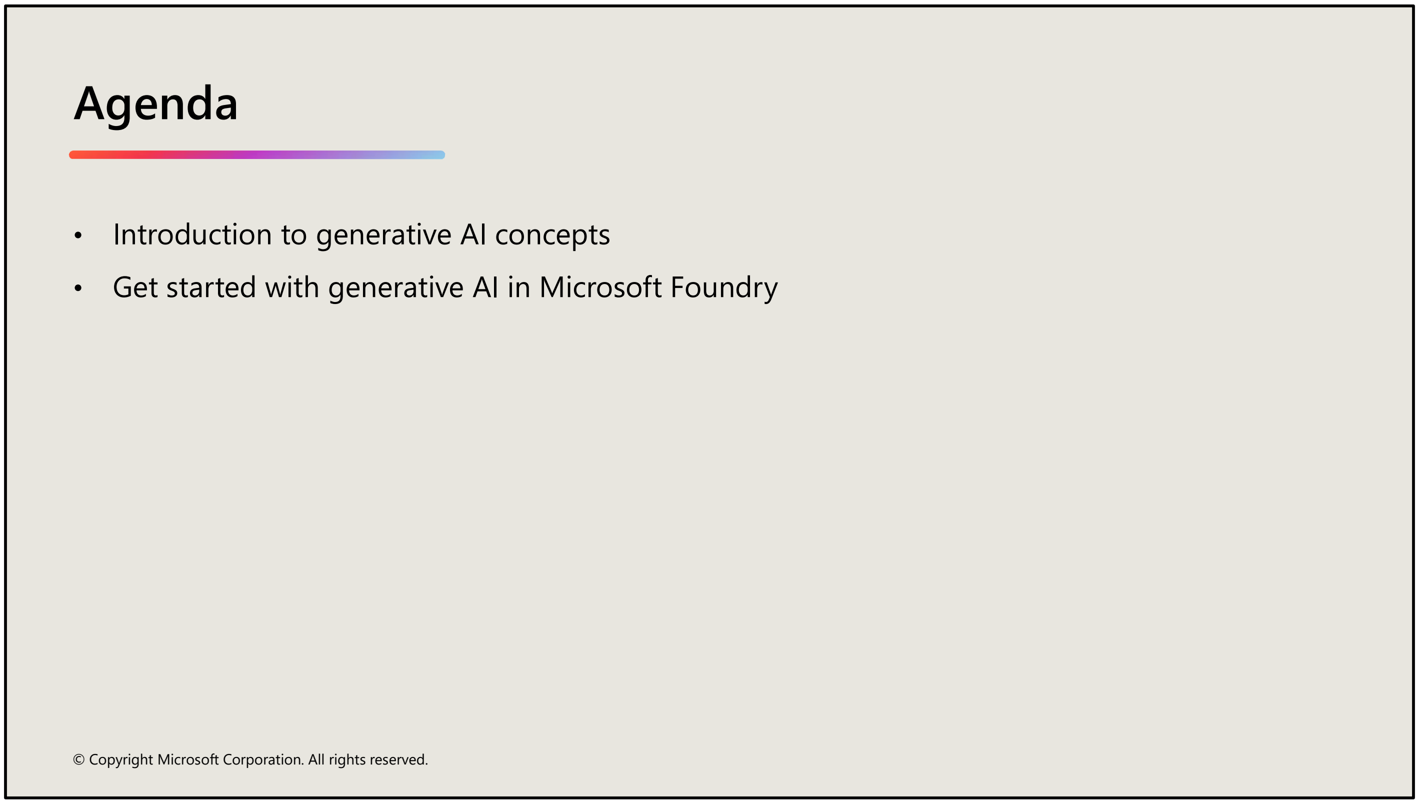
This content is meant to be taught in companion to:

- https://learn.microsoft.com/training/modules/fundamentals-generative-ai/
- https://learn.microsoft.com/training/modules/get-started-generative-ai-azure/



Generative AI and agents in context

In this section we will focus on generative AI and discuss agents and automation

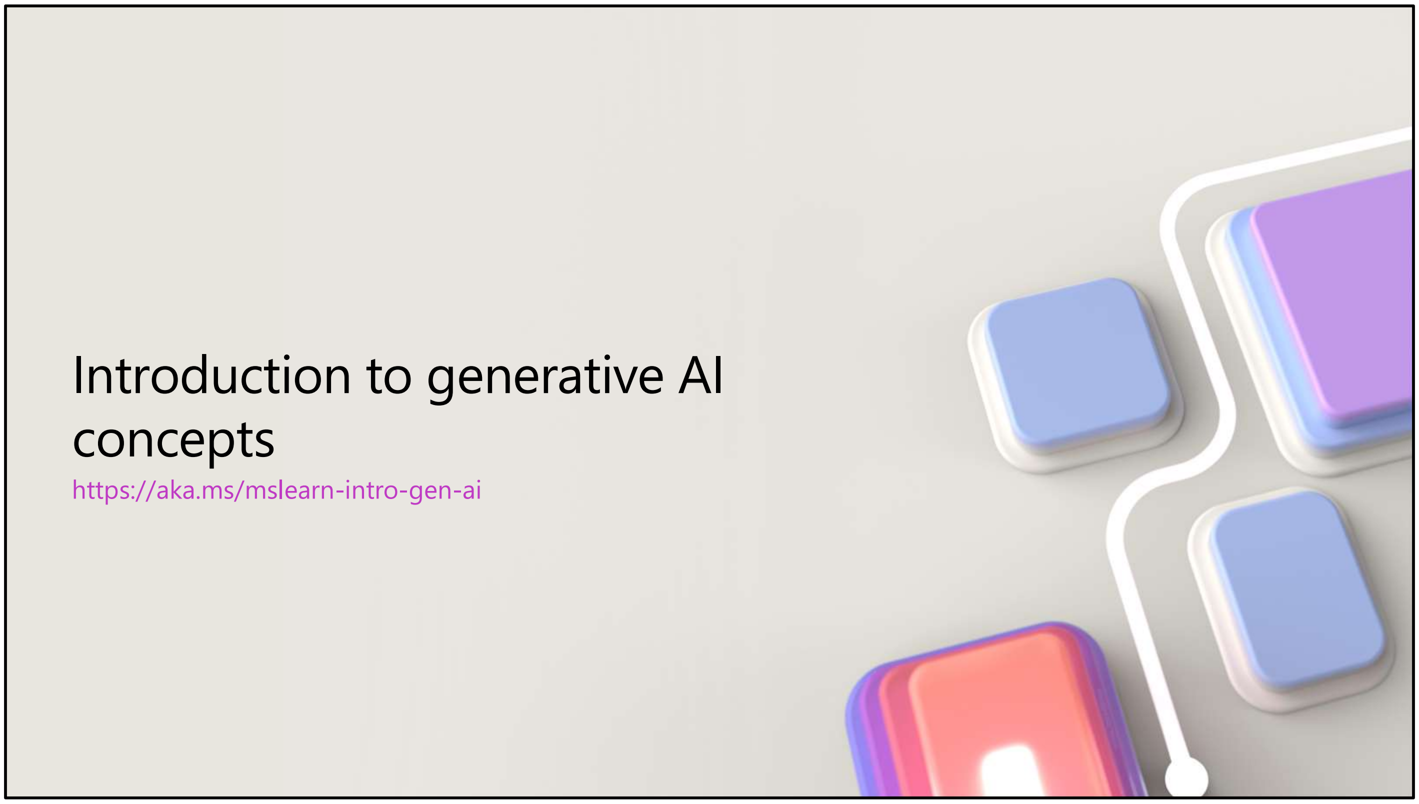


Agenda

- Introduction to generative AI concepts
- Get started with generative AI in Microsoft Foundry

Time estimates:

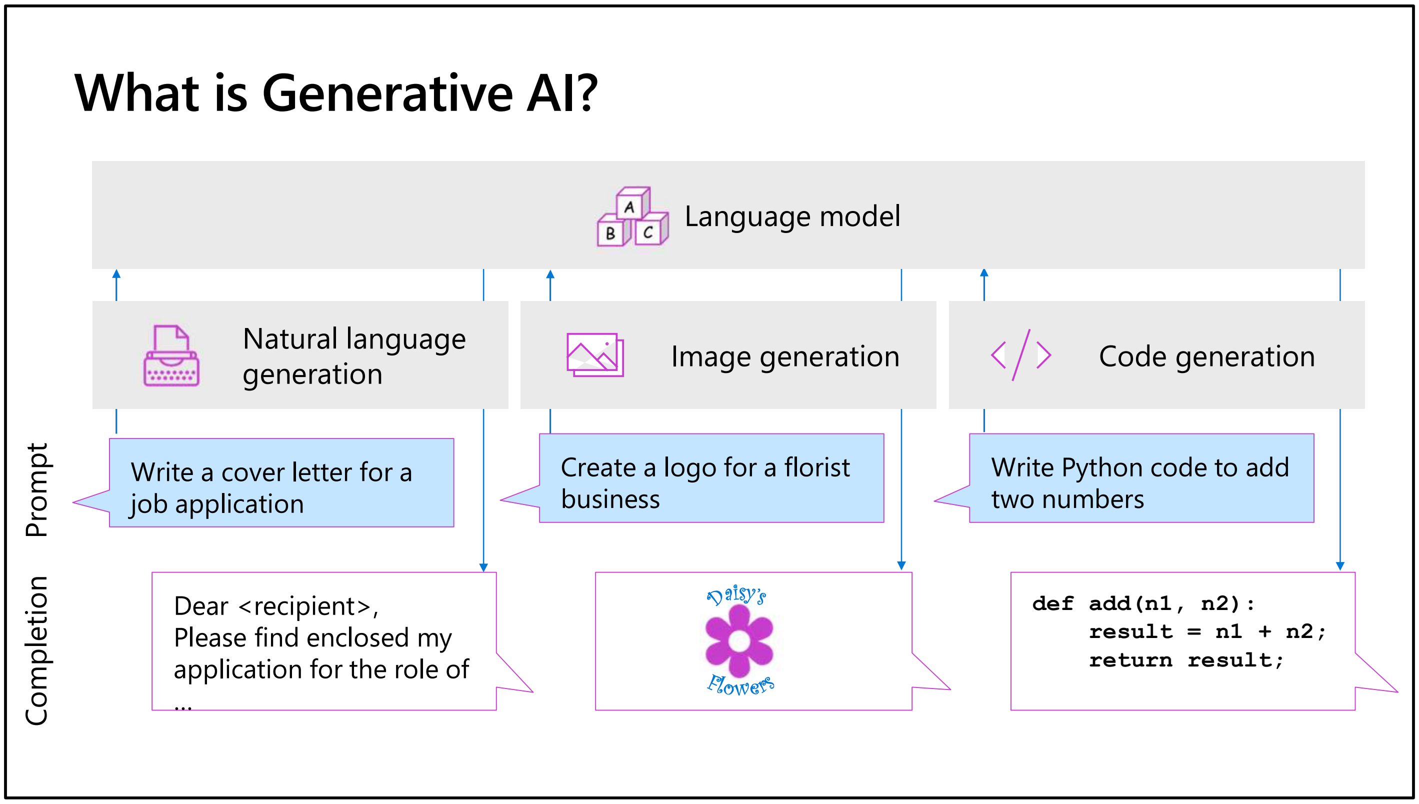
- Introduction to generative AI concepts – 40 mins (including lab exercise)
- Get started with generative AI in Microsoft Foundry – 35 mins (including demo)



Introduction to generative AI concepts

https://aka.ms/mslearn-intro-gen-ai

Use the link on the slide to see the Microsoft Learn learning module from which this section is derived.



What is Generative AI?

Language model
Natural language generation
Image generation
Code generation
Prompt
Completion
Write a cover letter for a job application
Create a logo for a florist business
Write Python code to add two numbers
Dear <recipient>, Please find enclosed my application for the role of …
Daisy's Flowers

```
def add(n1, n2):
    result = n1 + n2;
    return result;
```

Vad är Generativ AI?

Generativ AI är en typ av artificiell intelligens som kan **skapa innehåll** – till exempel text, bilder eller kod – utifrån instruktioner (prompter) från användaren. Den bygger på en **språkmodell**, som är en AI-modell tränad på enorma mängder data, ofta text från internet, böcker och kod. Den lär sig språkets mönster och kan sedan skapa nytt innehåll som liknar det den har lärt sig.

Bilden visar tre vanliga användningsområden:

1. Natural Language Generation (NLG)

Detta innebär att AI:n genererar **text** på naturligt språk.
Exempel i bilden:
📝 "*Write a cover letter for a job application.*"
AI:n svarar med en välformulerad ansökan som börjar med:
"*Dear , Please find enclosed my application...*"

Detta används i många sammanhang: CV, e-post, artiklar eller kundsupport.

2. Image Generation

Här skapar AI **bilder** från en textbeskrivning.
Exempel i bilden:
🎨 "*Create a logo for a florist business.*"
AI:n genererar automatiskt en logotyp med en blomma och texten "*Daisy's Flowers.*"

Denna teknik används för logotyper, illustrationer, designskisser och till och med konstverk.

3. Code Generation

AI:n kan också **skriva programkod**.
Exempel i bilden:
💻 "*Write Python code to add two numbers.*"
AI genererar korrekt kod:

def add(n1, n2): result = n1 + n2 return result Detta hjälper utvecklare att automatisera enklare koduppgifter eller snabbt få exempel.

Sammanfattning

Generativ AI handlar om att skapa något nytt – text, bild eller kod – baserat på vad användaren ber om. Det gör tekniken kraftfull inom många områden: kommunikation, design och programmering. Allt drivs av den underliggande **språkmodellen**, som lär sig av data och svarar på människors behov.

AI imitates human behavior by using machine learning to interact with the environment and execute tasks without explicit directions on what to output. *Generative AI* is a subset of AI in which an AI model creates original content in response to a natural language *prompt*. Generative AI applications take in natural language input, and return appropriate responses in a variety of formats:

Natural language generation

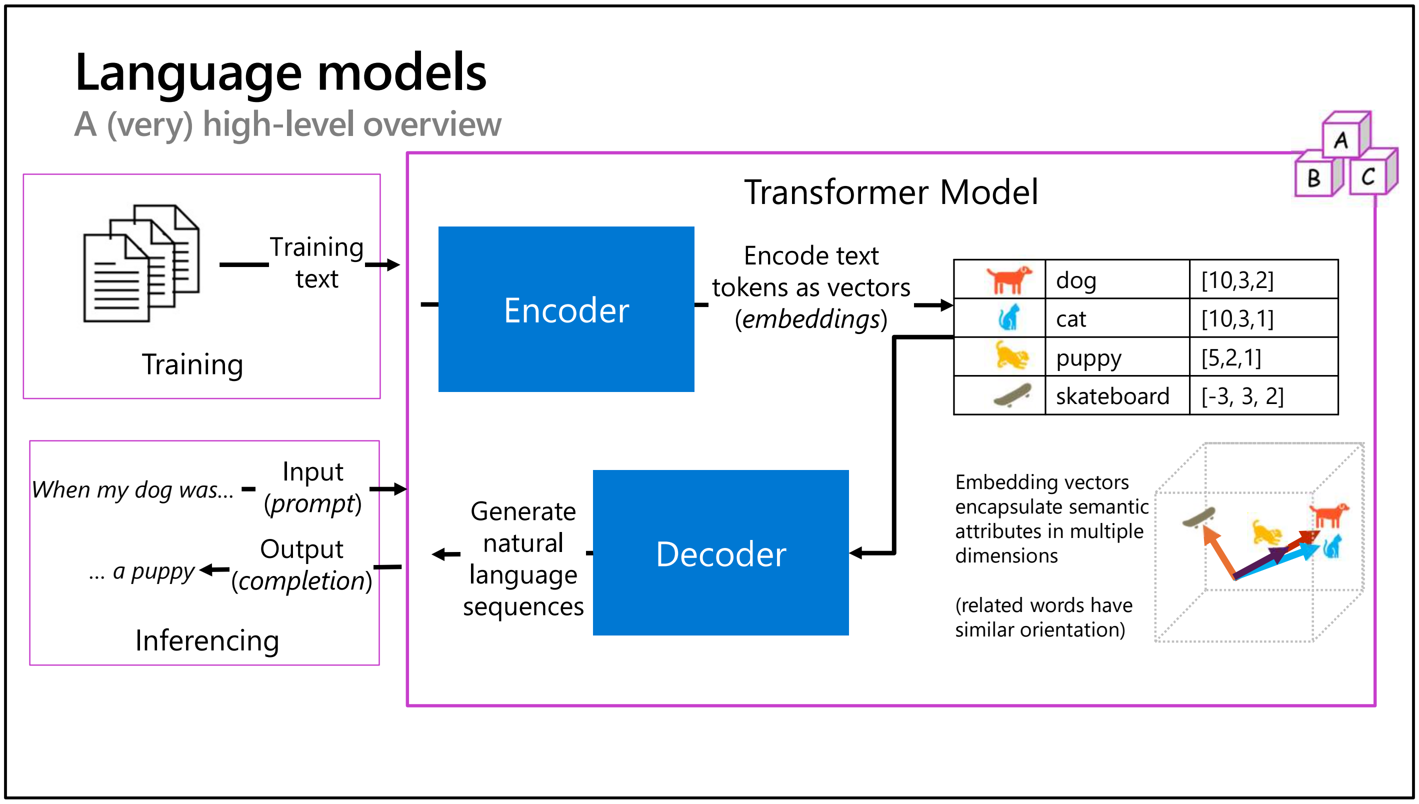
To generate a natural language response, you might submit a prompt such as "*Give me three ideas for a healthy breakfast including peppers.*" or *"Write a cover letter for my resume."*

Image generation

Some generative AI applications can interpret a natural language request and generate an appropriate image. For example, you might submit a request like "C*reate an image of an elephant eating a burger*" or "*Create a logo for a florist business*"

Code generation

Some generative AI applications are designed to help software developers write code. For example, you could submit a request like "*Show me how to code a game of tic-tac-toe with Python*" or *"Write Python code to add two numbers"*



Language models

A (very) high-level overview

Transformer Model
Training text
Training
Encoder
Encode text tokens as vectors (*embeddings*)
When my dog was…
Input (*prompt*)
Output (*completion*)
… a puppy
Inferencing
Generate natural language sequences
Decoder
Embedding vectors encapsulate semantic attributes in multiple dimensions
(related words have similar orientation)
A
B
C

	dog	[10,3,2]
	cat	[10,3,1]
	puppy	[5,2,1]
	skateboard	[-3, 3, 2]

🧠 **Vad är en språkmodell?**

En **språkmodell** är en AI som kan läsa, förstå och generera text – till exempel skriva meningar, svara på frågor eller föreslå nästa ord i en mening. Bilden visar hur en sådan modell, baserad på en **transformer**, fungerar på en övergripande nivå.

🔁 **Två faser: träning och användning**

1. Träning (Training)

Först tränas modellen på **massor av text**. Det kan vara böcker, webbsidor, artiklar – vilket kallas "*training text*".
Texten matas in i **encodern**, som gör om orden till **talrader** (vektorer), som representerar orden i matematisk form. Det kallas **embeddings**.

📦 **Exempel på embedding**

Orden "dog", "cat", och "puppy" får liknande vektorer:

dog → [10, 3, 2]

puppy → [5, 2, 1]

Eftersom orden är relaterade (de handlar om djur), får de **liknande riktning** i vektorrummet (se lilla 3D-boxen i bilden).

Ordet "skateboard" får däremot helt andra värden – eftersom det är något helt annat.

2. Användning (Inferencing)

När modellen är färdigtränad kan du använda den genom att ge den en **prompt** – ett påbörjat textstycke, till exempel:
"When my dog was..."

Denna input går återigen genom encodern, som kodar orden, och skickas sedan vidare till **decodern**.
Decodern **gissar fortsättningen**, i detta fall:
"...a puppy"

Modellen har alltså lärt sig språkets mönster och kan gissa vilket ord som troligen kommer härnäst.

✨ **Sammanfattning**

Encoder omvandlar ord till siffror (embeddings).

Decoder använder dessa för att skapa ny text.

När du ger modellen en mening, försöker den **förstå sammanhanget** och **fortsätta texten** på ett meningsfullt sätt.

Det är så språkmodeller som ChatGPT fungerar!

While the mathematical principles behind language models can be complex, a basic understanding of the architecture used to implement them can help you gain a conceptual understanding of how they work.

Today's cutting-edge large language models are based on the *transformer* architecture, which builds on and extends some techniques that have been proven successful in modeling vocabularies to support NLP tasks - and in particular in generating language. Transformer models are trained with large volumes of text, enabling them to represent the semantic relationships between words and use those relationships to predict probable sequences of text that make sense. Transformer models with a large enough vocabulary are capable of generating language responses that are tough to distinguish from human responses.

In practice, the specific implementations of the architecture vary – for example, the Bidirectional Encoder Representations from Transformers (BERT) model developed by Google to support their search engine uses only an *encoder* block to generate semantic vector representations of text, while the Generative Pretrained Transformer (GPT) model developed by OpenAI uses only a *decoder* block to generate sequences of natural language.

While a complete explanation of every aspect of transformer models is beyond the scope of this module, an explanation of some of the key elements in a transformer can help you get a sense for how they support generative AI.

The first step in training a transformer model is to decompose the training text into *tokens* - in other words, identify each unique text value. 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. With a sufficiently large set of training text, a vocabulary of many thousands of tokens could be compiled.

While it may be convenient to represent tokens as simple IDs - essentially creating an index for all the words in the vocabulary, they don't tell us anything about the meaning of the words, or the relationships between them. To create a vocabulary that encapsulates semantic relationships between the tokens, language models define contextual vectors, known as *embeddings*, for them. Vectors are multi-valued numeric representations of information, for example [10, 3, 1] in which each numeric element represents a particular attribute of the information. For language tokens, each element of a token's vector represents some semantic attribute of the token. The specific categories for the elements of the vectors in a language model are determined during training based on how commonly words are used together or in similar contexts.

Vectors represent lines in multidimensional space, describing *direction* and *distance* along multiple axes (you can impress your mathematician friends by calling these *amplitude* and *magnitude*). It can be useful to think of the elements in an embedding vector for a token as representing steps along a path in multidimensional space. For example, a vector with three elements represents a path in 3-dimensional space in which the element values indicate the units traveled forward/back, left/right, and up/down. Overall, the vector describes the direction and distance of the path from origin to end.

The elements of the tokens in the embeddings space each represent some semantic attribute of the token, so that semantically similar tokens should result in vectors that have a similar orientation – in other words they point in the same direction. a technique called *cosine similarity* is used to determine if two vectors have similar directions (regardless of distance), and therefore represent semantically linked words. For example, 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.

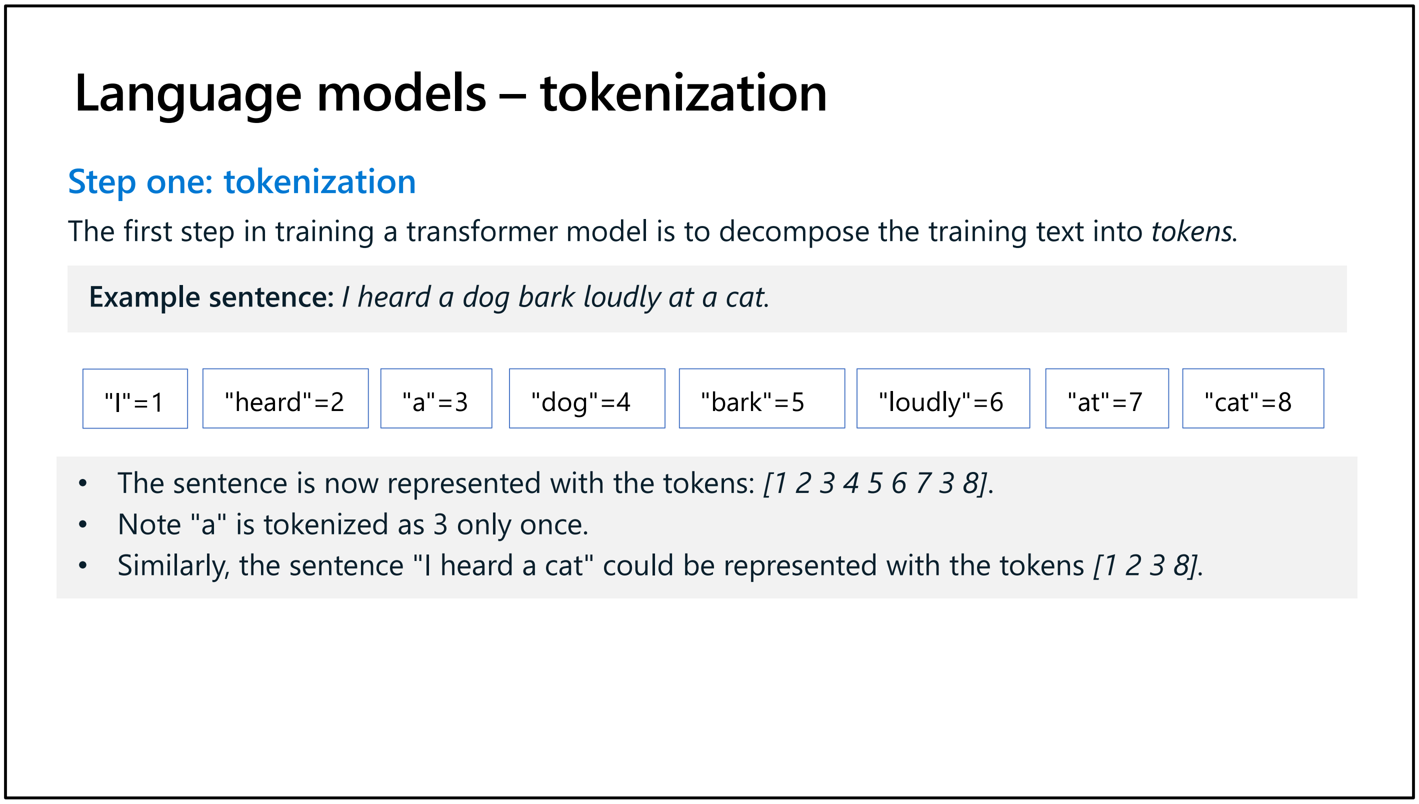
Note: The example shows a simple model in which each embedding has only three dimensions. Real language models have *many* more dimensions, which determines the number of *parameters* supported by the model.

The *encoder* and *decoder* blocks in a transformer model include multiple layers that form the neural network for the model. We don't need to go into the details of all these layers, but it's useful to consider one of the types of layers that is used in both blocks: *attention* layers. Attention is a technique used to examine a sequence of text tokens and try to quantify the strength of the relationships between them. In particular, *self-attention* involves considering how other tokens around one particular token influence that token's meaning.

In an encoder block, attention is used to examine each token in context, and determine an appropriate encoding for its vector embedding. The vector values are based on the relationship between the token and other tokens with which it frequently appears or which appear in similar contexts. This contextualized approach means that the same word might have multiple embeddings depending on the context in which it's used - for example "the bark of a tree" means something different to "I heard a dog bark."

In a decoder block, attention layers are used to predict the next token in a sequence. For each token generated, the model has an attention layer that takes into account the sequence of tokens up to that point. The model considers which of the tokens are the most influential when considering what the next token should be.

Remember that the attention layer is working with numeric vector representations of the tokens, not the actual text. In a decoder, the process starts with a sequence of token embeddings representing the text to be completed. During training, the goal is to predict the vector for the final token in the sequence based on the preceding tokens. The attention layer assigns a numeric *weight* to each token in the sequence so far. It uses that value to perform a calculation on the weighted vectors that produces an *attention score* that can be used to calculate a possible vector for the next token. In practice, a technique called *multi-head attention* uses different elements of the embeddings to calculate multiple attention scores. A neural network is then used to evaluate all possible tokens to determine the most probable token with which to continue the sequence. The process continues iteratively for each token in the sequence, with the output sequence so far being used regressively as the input for the next iteration – essentially building the output one token at a time.



Language models – tokenization

Step one: tokenization

The first step in training a transformer model is to decompose the training text into *tokens*.

Example sentence: *I heard a dog bark loudly at a cat.*

"I"=1	"heard"=2	"a"=3	"dog"=4	"bark"=5	"loudly"=6	"at"=7	"cat"=8

- The sentence is now represented with the tokens: *[1 2 3 4 5 6 7 3 8]*.
- Note "a" is tokenized as 3 only once.
- Similarly, the sentence "I heard a cat" could be represented with the tokens *[1 2 3 8]*.

Exempel: https://platform.openai.com/tokenizer

Token 1-100 längre ner bland kommentarer!

🧱 **Vad är tokenisering?**

Tokenisering är **det allra första steget** när en språkmodell (som t.ex. ChatGPT) ska läsa och förstå text.

Eftersom datorer inte förstår ord direkt, behöver orden först **översättas till siffror** – och det är precis vad tokenisering gör. Varje ord (eller del av ord) blir en **token**, som representeras av ett unikt nummer.

🔍 **Vad visar bilden?**

Meningen **"I heard a dog bark loudly at a cat."** delas upp i tokens:

Ord Token "I" 1 "heard" 2 "a" 3 "dog" 4 "bark" 5 "loudly" 6 "at" 7 "cat" 8 Så modellen representerar meningen som:
[1 2 3 4 5 6 7 3 8]
Lägg märke till att ordet **"a"** dyker upp två gånger men representeras av samma token (3).

🌍 **Olika språk – olika antal tokens**

Antalet tokens som behövs beror mycket på **språket** och **ordens längd och struktur**:

Engelska har ofta **många korta ord** → behöver färre tokens.
Exempel: "I love you" → 3 tokens.

Kinesiska och japanska har **kompakta tecken** → varje tecken blir ofta en egen token → fler tokens.
Exempel: "我爱你" ("I love you") → 3 tokens, men betyder samma sak.

Tyska eller finska har **långa sammansatta ord** → dessa kan behöva delas upp i flera tokens.
Exempel: "Krankenhausverwalter" → kanske 3 eller 4 tokens!

🧠 **Varför är detta viktigt?**

Modeller har ofta ett **begränsat antal tokens** de kan hantera per gång (t.ex. 4 000 eller 8 000 tokens). Ju fler tokens en text kräver, desto **dyrare och långsammare** blir bearbetningen – och ibland ryms inte all text i modellen.

✨ **Sammanfattning:**

Tokenisering översätter ord till siffror.

Samma ord får alltid samma token.

Olika språk kräver olika många tokens.

Tokenisering är nyckeln till att AI kan förstå och generera språk!

Token 1-100

1 = "!"

2 = "\""

3 = "#"

4 = "$"

5 = "%"

6 = "&"

7 = "'"

8 = "("

9 = ")"

10 = "*"

11 = "+"

12 = ","

13 = "-"

14 = "."

15 = "/"

16 = "0"

17 = "1"

18 = "2"

19 = "3"

20 = "4"

21 = "5"

22 = "6"

23 = "7"

24 = "8"

25 = "9"

26 = ":"

27 = ";"

28 = "<"

29 = "="

30 = ">"

31 = "?"

32 = "@"

33 = "A"

34 = "B"

35 = "C"

36 = "D"

```
37 = "E"
38 = "F"
39 = "G"
40 = "H"
41 = "I"
42 = "J"
43 = "K"
44 = "L"
45 = "M"
46 = "N"
47 = "O"
48 = "P"
49 = "Q"
50 = "R"
51 = "S"
52 = "T"
53 = "U"
54 = "V"
55 = "W"
56 = "X"
57 = "Y"
58 = "Z"
59 = "["
60 = "\\"
61 = "]"
62 = "^"
63 = "_"
64 = "`"
65 = "a"
66 = "b"
67 = "c"
68 = "d"
69 = "e"
70 = "f"
71 = "g"
72 = "h"
73 = "i"
74 = "j"
75 = "k"
76 = "l"
77 = "m"
78 = "n"
79 = "o"
80 = "p"
81 = "q"
82 = "r"
83 = "s"
```

84 = "t"
85 = "u"
86 = "v"
87 = "w"
88 = "x"
89 = "y"
90 = "z"
91 = "{"
92 = "|"
93 = "}"
94 = "~"
95 = "£"
96 = "¥"
97 = "§"
98 = "©"
99 = "«"
100 = "®"

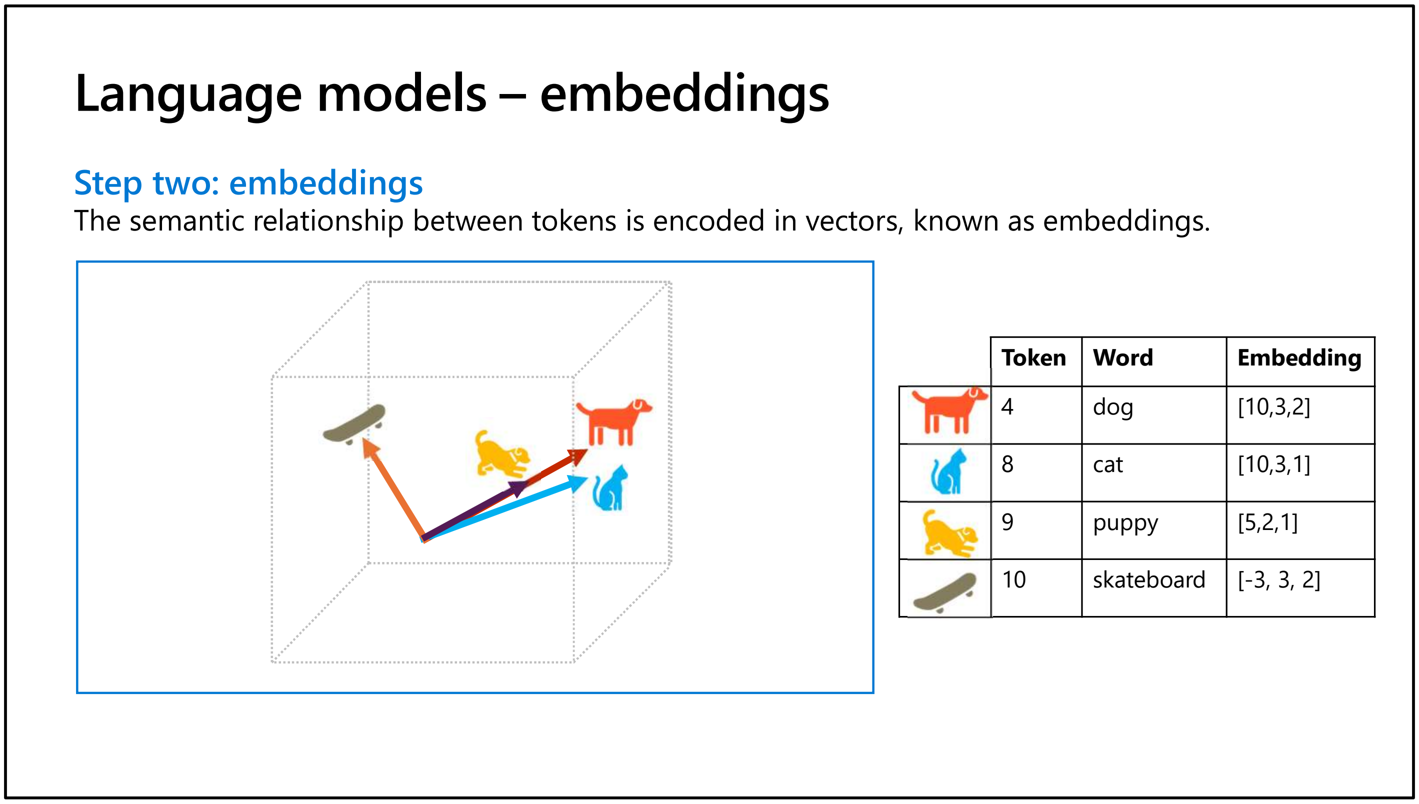
The first step in training a transformer model is to decompose the training text into *tokens* - in other words, identify each unique text value.

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.

As you continue to train the model, each new token in the training text is added to the vocabulary with appropriate token IDs:

- “I” is 1
- “heard” is 2
- “a” is 3
- “dog” is 4

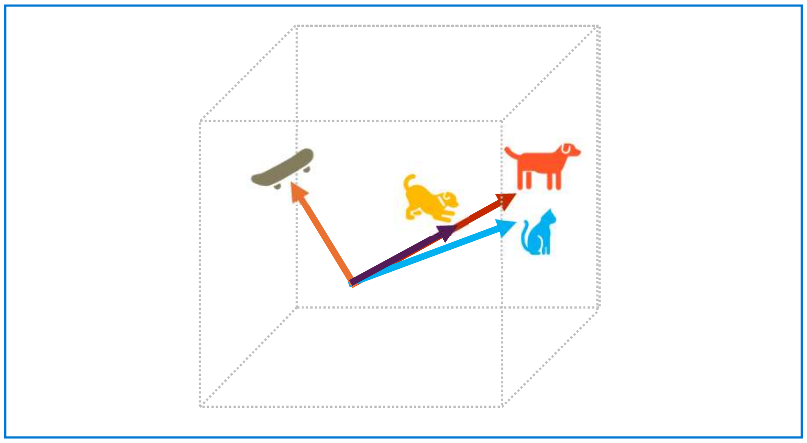
With a sufficiently large set of training text, a vocabulary of many thousands of tokens could be compiled.



Language models – embeddings

Step two: embeddings

The semantic relationship between tokens is encoded in vectors, known as embeddings.



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

Film om vektorer:

https://youtu.be/vr0MHUMZPv8

🧠 **Vad är embeddings?**

När en språkmodell (som ChatGPT) ska förstå ord behöver den ett smartare sätt än att bara ge varje ord ett nummer (token). Till exempel:

Ord Token dog 4 puppy 9 skateboard 10 Att bara ha siffror berättar inget om vad orden betyder eller om de liknar varandra. Orden är som ID-nummer – praktiska, men innehållslösa.

✨ **Därför används embeddings**

I stället för enkla nummer används **embeddings**, som är **vektorer** – alltså rader med siffror, till exempel:
dog → [10, 3, 2]
puppy → [5, 2, 1]

Dessa siffror beskriver **betydelsen** av ordet, fast i matematisk form. Varje siffra representerar ett visst "semantiskt drag" – alltså något som säger något om ordets användning eller mening.

🧭 **Vektorer i ett rum med flera dimensioner**

Du kan tänka dig en vektor som en **pil i ett rum**, där siffrorna visar hur långt den går åt olika håll.
Om vektorn har tre tal, som [10, 3, 2], rör den sig i ett **tredimensionellt rum** (fram/bak, höger/vänster, upp/ner).

Ju mer lika två pilar pekar, desto mer lika är orden i betydelse.
➡️ Därför pekar *dog* och *puppy* nästan åt samma håll.
➡️ *cat* pekar också i liknande riktning.
❌ Men *skateboard* pekar åt ett helt annat håll – det betyder något helt annat.

🧮 **Hur vet modellen vad siffrorna ska vara?**

När modellen tränas läser den **miljontals meningar** och lär sig vilka ord som ofta används tillsammans. På så sätt skapas vektorerna (embeddings) automatiskt.

Det finns olika tekniker för att skapa embeddings, till exempel:

Word2Vec

Transformermodellens encoder-del

🔍 **Bonus: Cosinuslikhet**

För att mäta hur lika två vektorer är använder man ofta **cosinuslikhet** – ett mått som säger om två vektorer pekar åt samma håll, även om de är olika långa. Detta är ett sätt att se om orden **betyder något liknande**, oavsett hur vanligt de är.

🧠 **Viktigt att veta!**

I exemplet används bara **3 dimensioner**, men i riktiga språkmodeller används **hundratals eller tusentals**.

Embeddings är centralt för att språkmodeller ska kunna förstå **betydelser och samband mellan ord** – inte bara känna igen stavningar.

🧾 **Sammanfattning:**

Ord görs först om till token-ID (t.ex. "dog" = 4).

Sedan omvandlas varje token till en **embedding-vektor** med siffror som beskriver betydelse.

Ord med liknande betydelse får vektorer som pekar i liknande riktning.

Detta gör att AI:n kan förstå hur ord hänger ihop – t.ex. att *puppy* liknar *dog*, men inte *skateboard*.

Det är så språkmodeller börjar förstå språk – med **vektorer som fångar mening**!

While it may be convenient to represent tokens as simple IDs - essentially creating an index for all the words in the vocabulary, they don't tell us anything about the meaning of the words, or the relationships between them. To create a vocabulary that encapsulates semantic relationships between the tokens, we define contextual vectors, known as *embeddings*, for them. Vectors are multi-valued numeric representations of information, for example [10, 3, 1] in which each numeric element represents a particular attribute of the information. For language tokens, each element of a token's vector represents some semantic attribute of the token. The specific categories for the elements of the vectors in a language model are determined during training based on how commonly words are used together or in similar contexts.

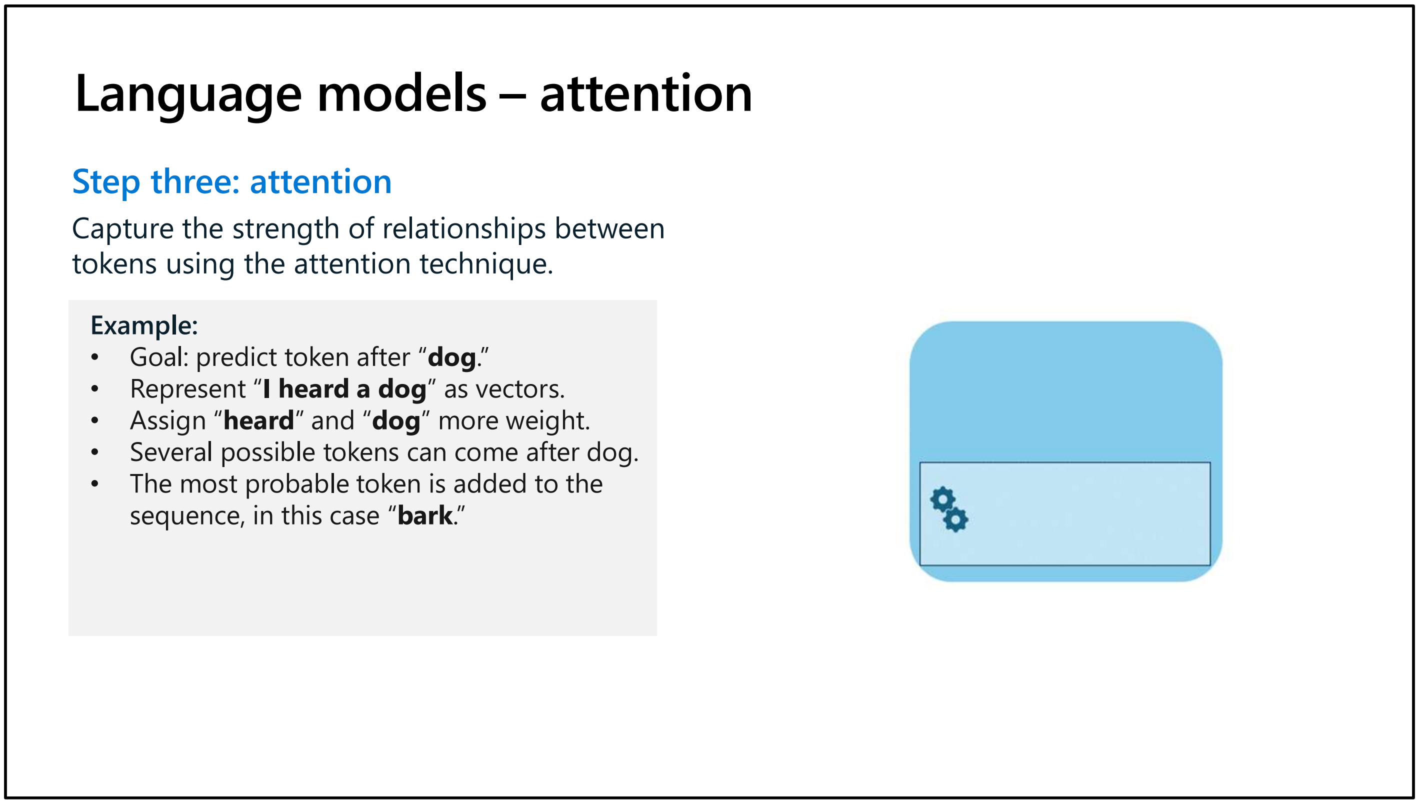
Vectors represent lines in multidimensional space, describing *direction* and *distance* along multiple axes (you can impress your mathematician friends by calling these *amplitude* and *magnitude*). It can be useful to think of the elements in an embedding vector for a token as representing steps along a path in multidimensional space. For example, a vector with three elements represents a path in 3-dimensional space in which the element values indicate the units traveled forward/back, left/right, and up/down. Overall, the vector describes the direction and distance of the path from origin to end.

The elements of the tokens in the embeddings space each represent some semantic attribute of the token, so that semantically similar tokens should result in vectors that have a similar orientation – in other words they point in the same direction. a technique called *cosine similarity* is used to determine if two vectors have similar directions (regardless of distance), and therefore represent semantically linked words. For example, 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.

Note:

The example shows a simple model in which each embedding has only three dimensions. Real language models have many more dimensions.

There are multiple ways you can calculate appropriate embeddings for a given set of tokens, including language modeling algorithms like *Word2Vec* or the *encoder* block in a transformer model.



Language models – attention

Step three: attention

Capture the strength of relationships between tokens using the attention technique.

Example:
- Goal: predict token after "**dog**."
- Represent "**I heard a dog**" as vectors.
- Assign "**heard**" and "**dog**" more weight.
- Several possible tokens can come after dog.
- The most probable token is added to the sequence, in this case "**bark**."

Film om attention:

https://youtu.be/oMeIDqRguLY

🧠 **Vad är "attention" i en språkmodell?**

När en AI läser text, måste den kunna förstå **vilka ord som är viktiga i en mening**. Det är här **attention-tekniken** kommer in – den hjälper modellen att fokusera på rätt ord, ungefär som vi människor gör när vi läser.

📌 **Exempel: "I heard a dog"**

Modellen ska nu förutspå vilket ord som ska komma efter "dog".
Det kan bli många olika saker – men om modellen fokuserar mer på orden "heard" och "dog", ökar chansen att den väljer ett passande ord som **"bark"**.

Detta kallas **self-attention** – modellen tittar tillbaka på tidigare ord och avgör vilka som är viktiga för att bestämma nästa ord.

🔧 **Hur fungerar det inuti modellen?**

En transformer-modell (som GPT-4) består av två delar:

Encoder – förstår inmatningen (t.ex. "I heard a dog").

Decoder – gissar vad som ska komma härnäst (t.ex. "bark").

Både encoder och decoder använder **attention-lager**.

🧩 **Encoder:**

Ser hela inmatningen.

Bestämmer hur orden påverkar varandra (t.ex. "bark" betyder något annat i "the tree's bark" än i "the dog barked").

Skapar "smarta" vektorer (embeddings) som fångar ordens **mening i sammanhanget**.

🎯 **Decoder:**
Skapar en mening, ett ord i taget.
Varje gång den gissar ett nytt ord, tittar den på tidigare ord och **räknar ut vilka som är viktigast**.
Ger olika **vikter** till varje tidigare ord – t.ex. kanske "dog" får högre vikt än "I".

🔁 **Hur det funkar i praktiken**
Alla ord görs om till **vektorer** (talrader).
Attention-lagret **räknar ut vikter** – alltså hur mycket varje tidigare ord påverkar nästa ord.
En **poäng (attention score)** beräknas för varje ord.
En **neural nätverksdel** använder poängen för att gissa nästa ord.
Det ordet läggs till sekvensen – och nästa ord förutsägs utifrån hela den nya meningen.
Detta sker **ett ord i taget**, och upprepas tills hela meningen är klar.

🧠 **Varför är detta så viktigt?**
Utan attention skulle modellen bara gissa nästa ord utifrån senaste token.
Med attention kan modellen tänka bredare och väga in fler ord bakåt i texten.

Exempel:
"I heard a dog" → hög sannolikhet för "bark"
"I saw a tree" → kanske "with thick bark"

Samma ord (**bark**) betyder något helt olika beroende på **sammanhang**.
Det är **attention** som gör det möjligt för modellen att förstå det!

✨ **Och varför fungerar det så bra?**
Modellen är tränad på **väldigt mycket text** – så den har sett många exempel på hur ord används tillsammans. Med hjälp av attention kan den använda den kunskapen för att skapa **meningsfull text**, ord för ord.

Det gör att den verkar **smart och mänsklig**, även om den egentligen bara gör matematiska beräkningar med stora mängder data.

The *encoder* and *decoder* blocks in a transformer model include multiple layers that form the neural network for the model. We don't need to go into the details of all these layers, but it's useful to consider one of the types of layers that is used in both blocks: *attention* layers. Attention is a technique used to examine a sequence of text tokens and try to quantify the strength of the relationships between them. In particular, *self-attention* involves considering how other tokens around one particular token influence that token's meaning.

In an encoder block, attention is used to examine each token in context, and determine an appropriate encoding for its vector embedding. The vector values are based on the relationship between the token and other tokens with which it frequently appears. This contextualized approach means that the same word might have multiple embeddings depending on the context in which it's used - for example "the bark of a tree" means something different to "I heard a dog bark."

In a decoder block, attention layers are used to predict the next token in a sequence. For each token generated, the model has an attention layer that takes into account the sequence of tokens up to that point. The model considers which of the tokens are the most influential when considering what the next token should be.

Remember that the attention layer is working with numeric vector representations of the tokens, not the actual text. In a decoder, the process starts with a sequence of token embeddings representing the text to be completed.

During training, the goal is to predict the vector for the final token in the sequence based on the preceding tokens. The attention layer assigns a numeric *weight* to each token in the sequence so far. It uses that value to perform a calculation on the weighted vectors that produces an *attention score* that can be used to calculate a possible vector for the next token. In practice, a technique called *multi-head attention* uses different elements of the embeddings to calculate multiple attention scores. A neural network is then used to evaluate all possible tokens to determine the most probable token with which to continue the sequence. The process continues iteratively for each token in the sequence, with the output sequence so far being used regressively as the input for the next iteration – essentially building the output one token at a time.

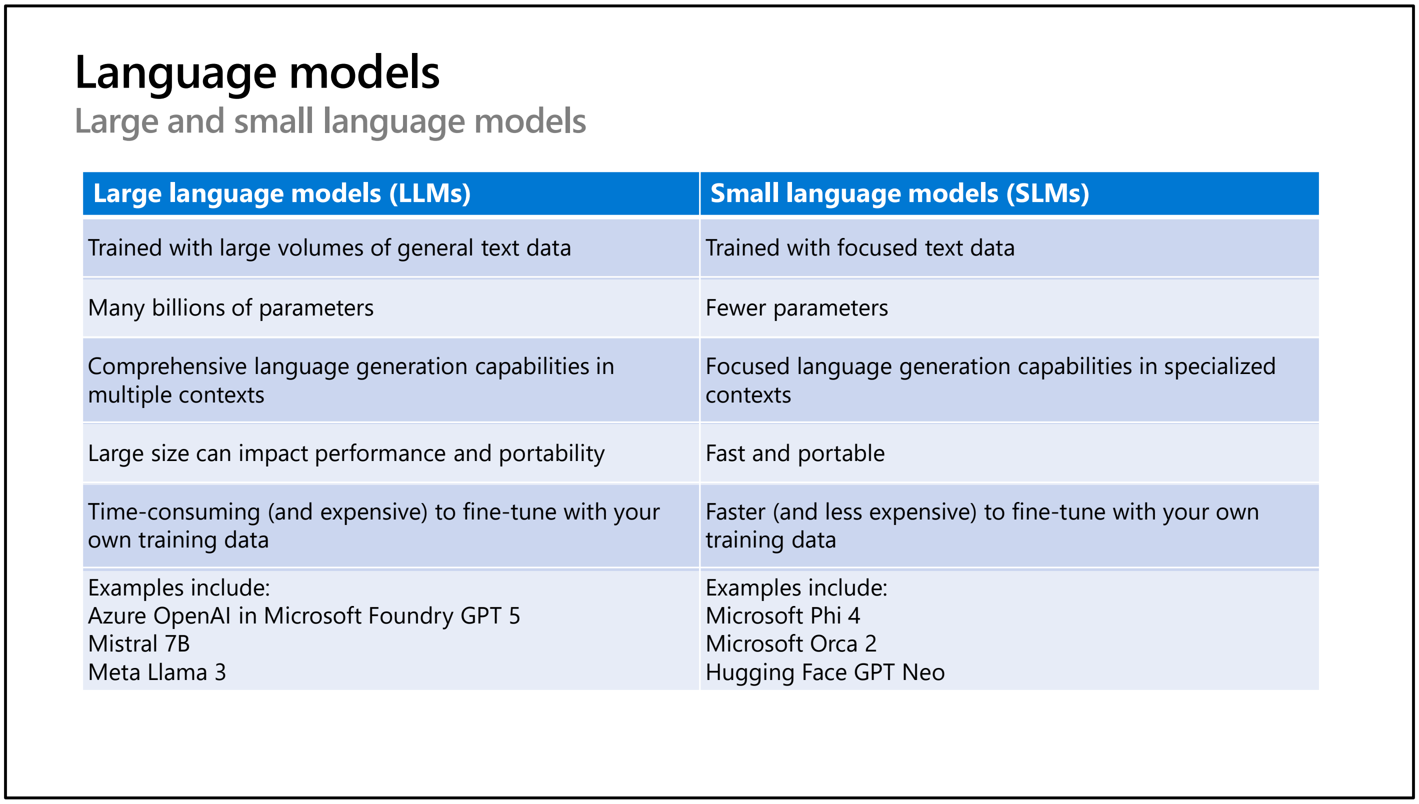
The following animation shows a simplified representation of how this works – in reality, the calculations performed by the attention layer are more complex; but the principles can be simplified as shown:

1. A sequence of token embeddings is fed into the attention layer. Each token is represented as a vector of numeric values.
2. The goal in a decoder is to predict the next token in the sequence, which will also be a vector that aligns to an embedding in the model's vocabulary.
3. The attention layer evaluates the sequence so far and assigns weights to each token to represent their relative influence on the next token.
4. The weights can be used to compute a new vector for the next token with an attention score. Multi-head attention uses different elements in the embeddings to calculate multiple alternative tokens.
5. A fully connected neural network uses the scores in the calculated vectors to predict the most probable token from the entire vocabulary.
6. The predicted output is appended to the sequence so far, which is used as the input for the next iteration.

During training, the actual sequence of tokens is known – we just mask the ones that come later in the sequence than the token position currently being considered. As in any neural network, the predicted value for the token vector is compared to the actual value of the next vector in the sequence, and the loss is calculated. The weights are then incrementally adjusted to reduce the loss and improve the model. When used for inferencing (predicting a new sequence of tokens), the trained attention layer applies weights that predict the most probable token in the model's vocabulary that is semantically aligned to the sequence so far.

What all of this means, is that a transformer model such as GPT-4 (the model behind ChatGPT and Bing) is designed to take in a text input (called a prompt) and generate a syntactically correct output (called a completion). In effect, the "magic" of the model is that it has the ability to string a coherent sentence together. This ability doesn't imply any "knowledge" or "intelligence" on the part of the model; just a large vocabulary and the ability to generate meaningful sequences of words. What makes a large language model like GPT-4 so powerful however, is the sheer volume of data with which it has been trained (public and licensed data from the Internet) and the complexity of the network. This enables the model to generate completions that are based on the relationships between words in the vocabulary on which the model was trained; often generating output that is indistinguishable from a human response to the same prompt.

A diagram that shows how the attention technique helps the large language model identify the next word in the sequence. The image shows words and their positional encoding layer coordinates.



Language models

Large and small language models

Large language models (LLMs)	Small language models (SLMs)
Trained with large volumes of general text data	Trained with focused text data
Many billions of parameters	Fewer parameters
Comprehensive language generation capabilities in multiple contexts	Focused language generation capabilities in specialized contexts
Large size can impact performance and portability	Fast and portable
Time-consuming (and expensive) to fine-tune with your own training data	Faster (and less expensive) to fine-tune with your own training data
Examples include: Azure OpenAI in Microsoft Foundry GPT 5 Mistral 7B Meta Llama 3	Examples include: Microsoft Phi 4 Microsoft Orca 2 Hugging Face GPT Neo

🧠 **Vad visar bilden?**

Den jämför **Large Language Models (LLMs)** och **Small Language Models (SLMs)** – två olika typer av AI-språkmodeller.

LLMs (Stora språkmodeller) SLMs (Små språkmodeller) Tränade på **mycket text** från många områden Tränade på **fokuserad text**, ofta för specifika ämnen Har **miljarder parametrar** – alltså väldigt komplexa Har färre parametrar – lättare och mindre Kan hantera **många olika typer av uppgifter** Bäst på **specifika områden** (t.ex. teknisk support, juridik) Kan vara **tunga, långsamma och svåra att flytta Snabba och portabla**, passar bra för edge- och mobilanvändning Tar **mycket tid och pengar att anpassa (fine-tune)** Går **snabbt och billigt att träna vidare på** egen data **Exempel**: GPT-4.1, Mistral 7B, Meta Llama 3 **Exempel**: Microsoft Phi-3, Hugging Face GPT Neo

🛠️ **Hur gör man en egen SLM?**

Att skapa en egen **Small Language Model** är enklare och billigare än att träna ett LLM. Här är huvudstegen:

1. Välj en grundmodell (pre-trained SLM)

Exempel:

🤗 Hugging Face: GPT-Neo, DistilGPT2, TinyLlama

🧠 Microsoft: Phi-3 (finns som open source)

Du behöver **inte börja från noll**, utan tar en färdig modell som redan kan språk.

2. Samla din träningsdata

Texter inom **ett specifikt ämne** – t.ex. sjukvård, ekonomi, supportmanualer.

Formatet är ofta textfiler eller JSON.

3. Finetuna modellen
Du "lär om" modellen med din data så att den blir expert på just det innehållet.
Verktyg du kan använda:

transformers-biblioteket från Hugging Face

PEFT eller LoRA för lättviktig finetuning

Körs ofta i Google Colab, på en lokal GPU eller i molnet (t.ex. Azure)

4. Testa modellen
Be den svara på exempel.

Jämför med en vanlig LLM och se om din modell är **snabbare och mer träffsäker** inom sitt område.

5. Använd i appar eller offline
Eftersom SLM:er är små, kan du:

Köra dem på en **laptop**, **mobil**, eller **på en edge-enhet** (t.ex. Raspberry Pi).

Slippa molnberoende!

💡 **När ska man välja en SLM?**
✅ När du:

Behöver AI för **ett specifikt ämne**

Vill köra lokalt (snabbt, privat)

Har **liten budget**

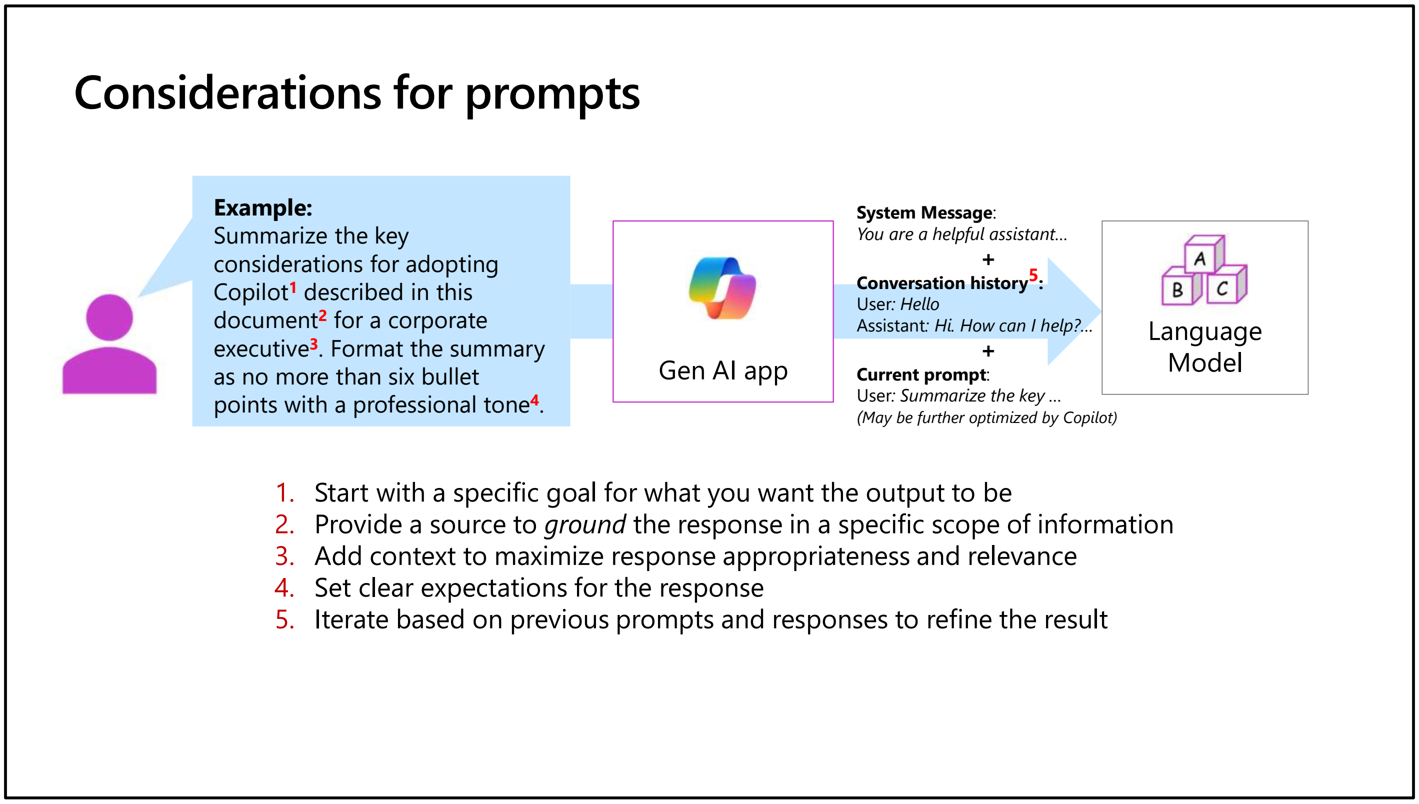
Vill ha **kontroll över träningen och datan**

Exempel:
Ett mindre företag som säljer reservdelar för cyklar vill erbjuda en chattbot på sin webbplats. Istället för att använda en stor och dyr språkmodell, tränar de en **Small Language Model (SLM)** på sina egna produktbeskrivningar, manualer och vanliga kundfrågor. Modellen blir snabb, billig att köra lokalt och mycket träffsäker – den förstår direkt vad "navkonvertering" eller "drevbyte" betyder. Resultatet blir en **snabb och användbar AI-assistent** som fungerar utan internet och ger kunderna snabba svar, utan att behöva gissa inom allmänna ämnen.

There are many language models available that you can use to power generative AI applications. In general, language models can be considered in two categorize: *Large Language Models* (LLMs) and *Small Language models* (SLMs).

LLMs are trained with vast quantities of text that represents a wide range of general subject matter – typically by sourcing data from the Internet and other generally available publications. When trained, LLMs have many billions (even trillions) of parameters (weights that can be applied to vector embeddings to calculate predicted token sequences), enabling them to exhibit comprehensive language generation capabilities in a wide range of conversational contexts. However, their large size can impact their performance and make them difficult to deploy locally on devices and computers. Additionally, if you want to *fine-tune* the model with additional data to customize its conversational subject expertise, the process can be time-consuming, and expensive in terms of the compute power required to perform the additional training.

Conversely, SLMs are trained with smaller, more subject-focused datasets and typically have fewer parameters than LLMs. This focused vocabulary makes them very effective in specific conversational topics, but less effective at more general language generation. The smaller size of SLMs can provide more options for deployment, including local deployment to devices and on-premises computers; and makes them faster and easier to fine-tune.



Considerations for prompts

Example:
Summarize the key considerations for adopting Copilot[1] described in this document[2] for a corporate executive[3]. Format the summary as no more than six bullet points with a professional tone[4].

Gen AI app

System Message:
You are a helpful assistant…
+
Conversation history[5]:
User: *Hello*
Assistant: *Hi. How can I help?…*
+
Current prompt:
User: *Summarize the key …*
(May be further optimized by Copilot)

Language Model

1. Start with a specific goal for what you want the output to be
2. Provide a source to *ground* the response in a specific scope of information
3. Add context to maximize response appropriateness and relevance
4. Set clear expectations for the response
5. Iterate based on previous prompts and responses to refine the result

Den här bilden handlar om hur man formulerar bra prompts (instruktioner) till AI-assistenter, som Copilot. För att få relevanta och användbara svar är det viktigt att tänka igenom hur man skriver sin fråga.

Bilden visar ett exempel på en bra prompt:

"Sammanfatta de viktigaste övervägandena för att införa Copilot som beskrivs i detta dokument, riktat till en företagsledare. Formatera sammanfattningen med högst sex punkter och en professionell ton."

Bakom detta exempel ligger fem nyckelprinciper:

Tydligt mål – Tala om exakt vad du vill att assistenten ska göra.

Källa – Ange var informationen finns (t.ex. ett dokument) så att svaret hålls inom rätt ramar.

Sammanhang – Beskriv för vem svaret är, vilket hjälper modellen att anpassa tonen.

Förväntningar – Ange form, stil eller längd (t.ex. punktform, max 6 punkter).

Iterera – Förbättra prompten efterhand, baserat på tidigare svar och behov.

Dessutom använder AI:n inte bara din aktuella prompt, utan också tidigare konversation ("conversation history") och en inbyggd systeminstruktion som säger t.ex. "Du är en hjälpsam assistent".

Sammanfattningsvis: Ju tydligare och mer kontextuell prompt du skriver, desto bättre blir svaret. Det är som att prata med en mycket duktig men bokstavstrogen medarbetare – du måste vara tydlig!

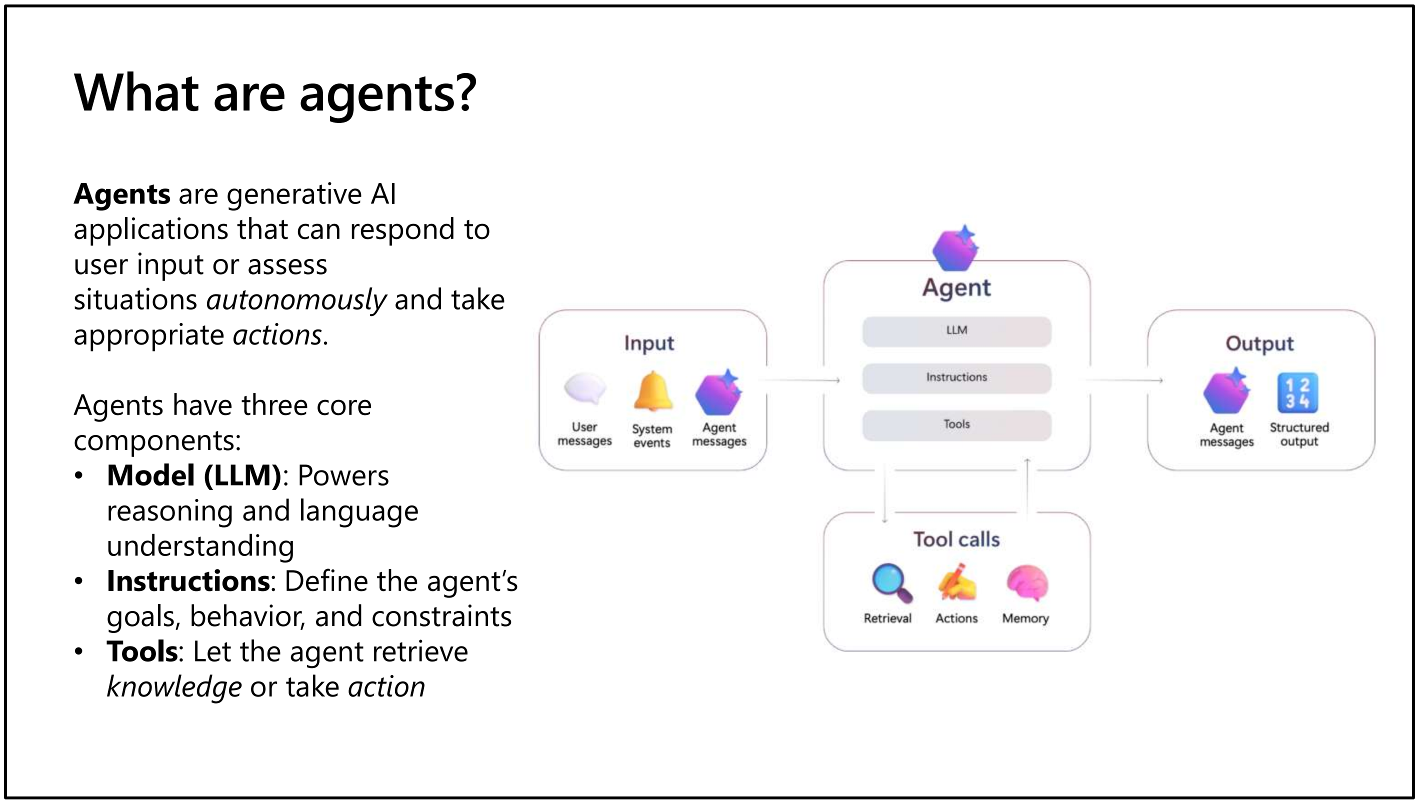
While some generative AI assistants provide buttons and other visual tools to interact with a language model, often you will use an assistant by typing (or speaking) a natural language prompt. The specific prompt that will yield the most effective results can vary depending on the specific task you're trying to accomplish, the assistant being used, and the language model it uses to support generative AI. However, there are some common prompting techniques you can apply to get the best out of your assistant:

1. Start with a specific goal for what you want the assistant to do – be explicit
2. Provide a source to *ground* the response in a specific scope of information -this will help ensure the response is based in real data
3. Add context – this can help the language model craft an appropriate response
4. Set clear expectations for the response – be explicit about the format and scope of the expected results
5. Iterate based on your past prompts and responses to refine the output

In most cases, an assistant does not just send your prompt as-is to the language model. Usually, your prompt is augmented with:

- A system message that sets conditions and constraints for the language model behavior.
- The conversation history for the current session, including past prompts and responses – this enables you to refine the response iteratively while maintaining the context of the conversation
- The current prompt – potentially optimized by the agent to reword it appropriately for the model or to add additional grounding data.to scope the response.

The success you have with assistants (and generative AI in general) depends largely on how well you craft your prompts. The importance of using good prompts has led to a new discipline of *prompt engineering*.

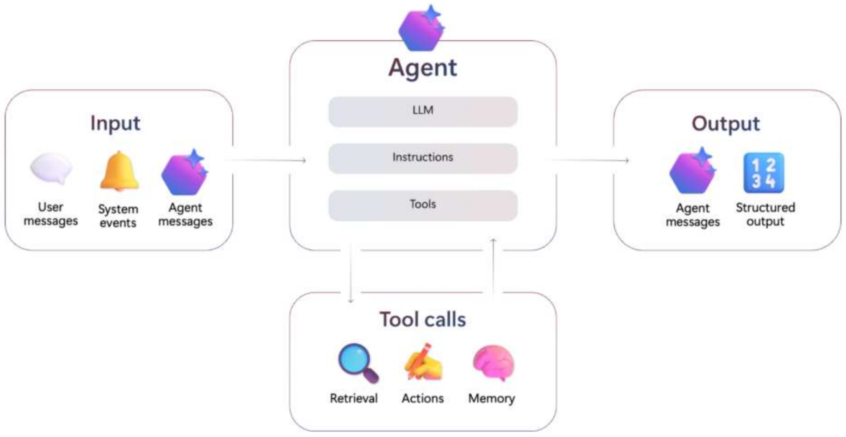


What are agents?

Agents are generative AI applications that can respond to user input or assess situations *autonomously* and take appropriate *actions*.

Agents have three core components:

- **Model (LLM)**: Powers reasoning and language understanding
- **Instructions**: Define the agent's goals, behavior, and constraints
- **Tools**: Let the agent retrieve *knowledge* or take *action*



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.

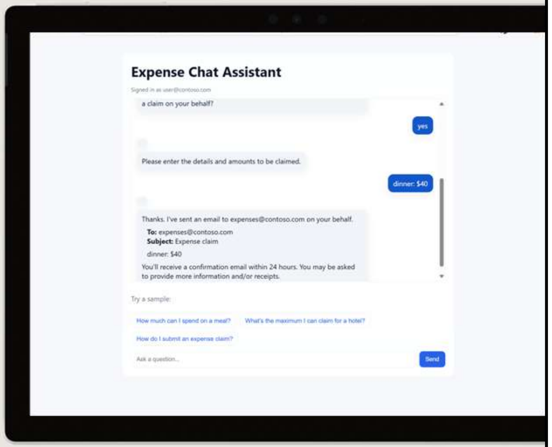
Today's AI solutions often contain a combination of assistant, agentic, and other AI capabilities. The process of coordinating and managing multiple AI components—such as models, data sources, tools, and workflows—to work together efficiently in a unified solution is known as *orchestration*.

Exercise

Explore generative AI agent scenarios

In this exercise, you'll explore generative AI apps and agents.

Start the exercise at:
https://go.microsoft.com/fwlink/?linkid=2334224



Expense Chat Assistant

Signed in as user@contoso.com

a claim on your behalf?

yes

Please enter the details and amounts to be claimed.

dinner: $40

Thanks. I've sent an email to expenses@contoso.com on your behalf.

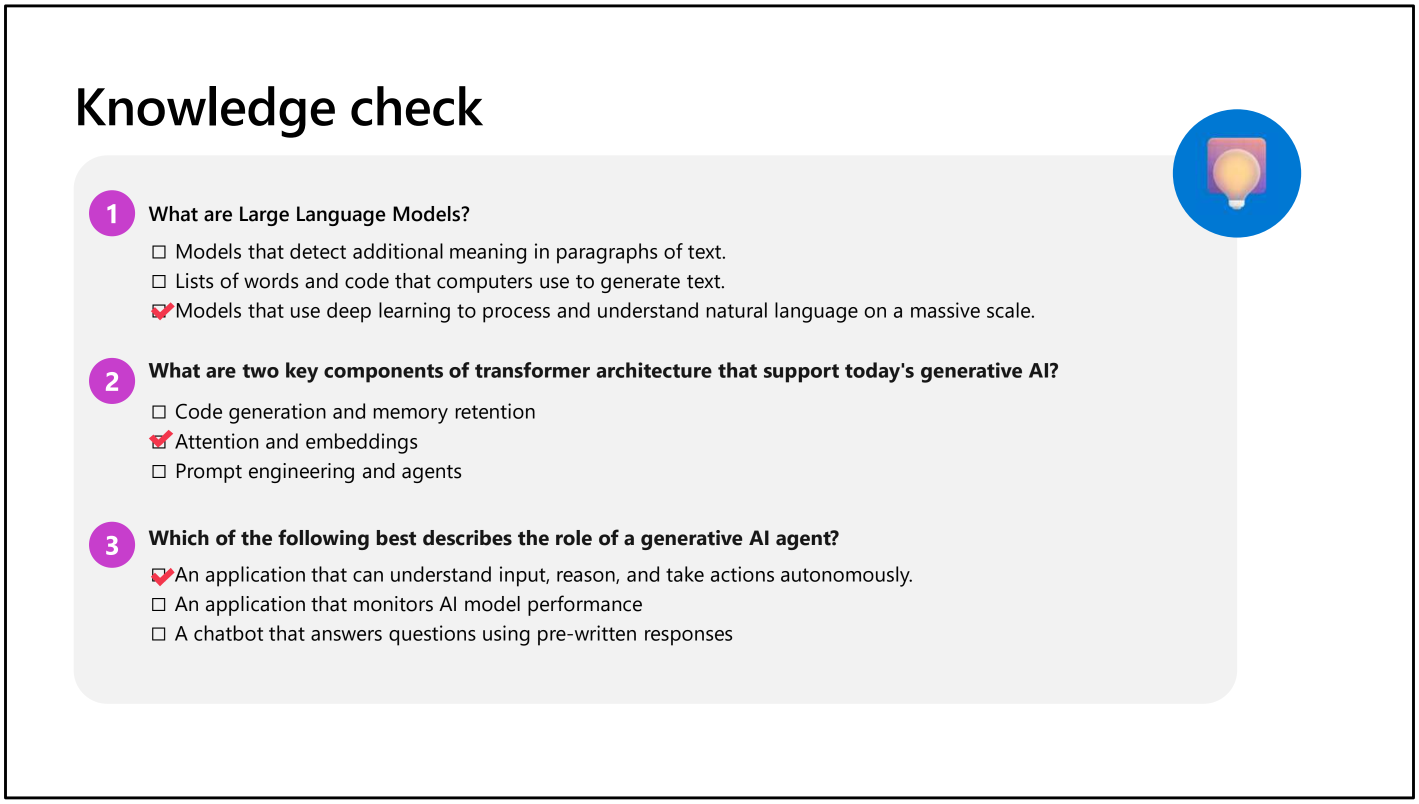
To: expenses@contoso.com
Subject: Expense claim

dinner: $40

You'll receive a confirmation email within 24 hours. You may be asked to provide more information and/or receipts.

Try a sample:
How much can I spend on a meal?
What's the maximum I can claim for a hotel?
How do I submit an expense claim?
Ask a question...
Send

Note: The applications used in this exercise are *simulations* - there are no actual generative AI models behind them and the capabilities have been simplified. However, they're based on real capabilities that you can implement with Microsoft Foundry Agent Service.



Knowledge check

1 What are Large Language Models?

☐ Models that detect additional meaning in paragraphs of text.
☐ Lists of words and code that computers use to generate text.
[✓ Models that use deep learning to process and understand natural language on a massive scale.]

2 **What are two key components of transformer architecture that support today's generative AI?**

☐ Code generation and memory retention
[✓ Attention and embeddings]
☐ Prompt engineering and agents

3 **Which of the following best describes the role of a generative AI agent?**

[✓ An application that can understand input, reason, and take actions autonomously.]
☐ An application that monitors AI model performance
☐ A chatbot that answers questions using pre-written responses

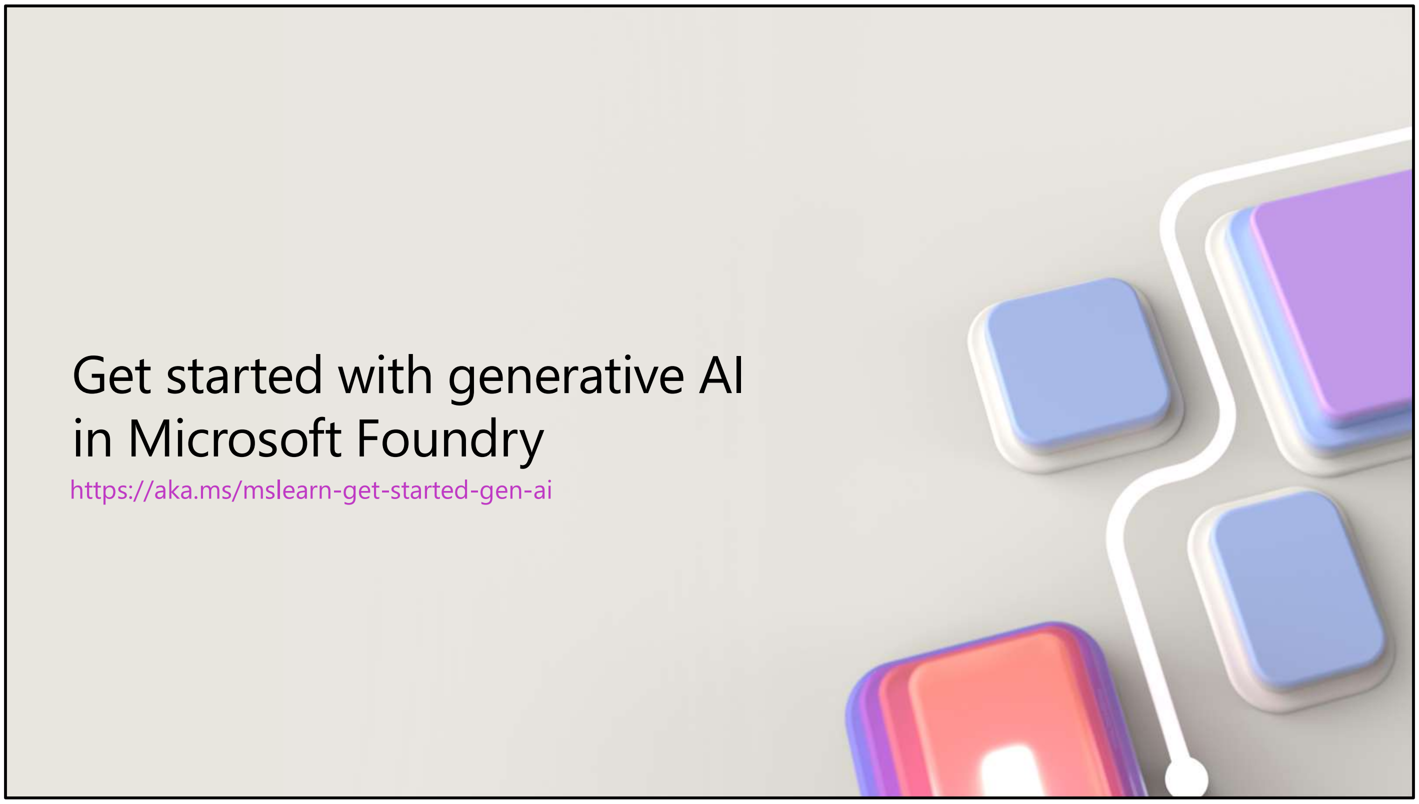
Här är korta svar med **max 10 ord per motivering**:

1. What are Large Language Models?
❌ **Fel:** Beskriver textanalys, inte språkgenerering i stor skala.
❌ **Fel:** LLM är inte statiska listor eller regler.
✅ **Rätt:** Använder djupinlärning för språkförståelse i stor skala.

2. Två nyckelkomponenter i transformer-arkitektur
❌ **Fel:** Inte grundläggande arkitekturkomponenter.
✅ **Rätt:** Centrala mekanismer för kontext och språkrepräsentation.
❌ **Fel:** Bygger på modeller, inte arkitekturens kärna.

3. Rollen för en generativ AI-agent
✅ **Rätt:** Kan tolka input, resonera och agera självständigt.
❌ **Fel:** Begränsat till övervakning, inte autonomt agerande.
❌ **Fel:** Saknar resonemang och autonomi.

Allow students a few minutes to think about the questions, and then reveal the correct answers.



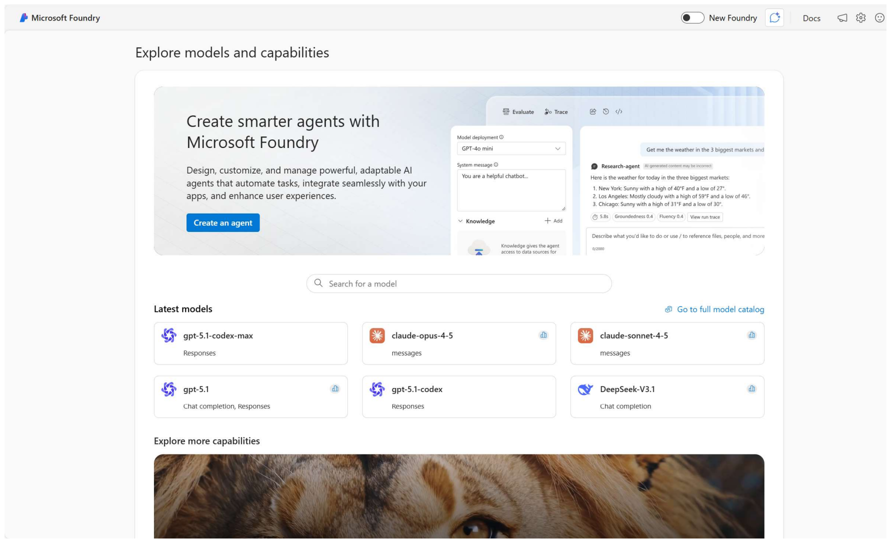
Get started with generative AI in Microsoft Foundry

https://aka.ms/mslearn-get-started-gen-ai

Use the link on the slide to see the Microsoft Learn learning module from which this section is derived.

What is Microsoft Foundry?

- A cloud-based platform for AI development
- Provides a collaborative pro-code environment with enterprise-grade security for AI innovation
- Use to explore, build, test, and deploy AI solutions that are reliable, scalable, and secure



Create smarter agents with Microsoft Foundry

Design, customize, and manage powerful, adaptable AI agents that automate tasks, integrate seamlessly with your apps, and enhance user experiences.

Create an agent

Get me the weather in the 3 biggest markets and

Here is the weather for today in the three biggest markets:
1. New York: Sunny with a high of 40°F and a low of 27°.
2. Los Angeles: Mostly cloudy with a high of 59°F and a low of 46°.
3. Chicago: Sunny with a high of 31°F and a low of 30°.

Latest models

gpt-5.1-codex-max	Responses
claude-opus-4-5	messages
claude-sonnet-4-5	messages
gpt-5.1	Chat completion, Responses
gpt-5.1-codex	Responses
DeepSeek-V3.1	Chat completion

Azure AI Foundry är en plattformstjänst (PaaS) för utvecklare som vill bygga kraftfulla och säkra AI-lösningar i stor skala. Den erbjuder en avancerad pro-code-miljö där team kan samarbeta med högsta säkerhetsnivå, särskilt anpassad för AI-innovation inom företag. Plattformen gör det möjligt att snabbt utforska, designa, testa och driftsätta AI-assistenter och applikationer med fokus på skalbarhet och ansvarsfull AI-användning.

Azure AI Foundry fungerar som en navpunkt för Microsofts bredare AI-ekosystem. Den integreras smidigt med andra verktyg som GitHub, Visual Studio och Microsoft Copilot Studio, vilket ger utvecklare flexibilitet att använda sina befintliga arbetsflöden.
Man kan också kombinera AI Foundry med Microsoft Fabric för datahantering och analys, samt med Azure OpenAI Service för tillgång till kraftfulla språkmodeller som GPT-4.

Till skillnad från lågkodplattformar, som Power Platform, riktar sig AI Foundry till professionella utvecklare som vill ha full kontroll över sin kod och sina AI-agenter. Den lämpar sig särskilt väl för organisationer som vill bygga och hantera generativa AI-tjänster internt – med robust testning, ansvarstagande och säkerhetsprinciper som en naturlig del av utvecklingsprocessen.

En garvad AI-utvecklare skulle sannolikt säga något i stil med:

"Azure AI Foundry är ett välkommet steg från Microsoft – äntligen en plattform som kombinerar enterprise-grade säkerhet med utvecklarvänliga verktyg på riktigt. Den är inte för 'low-code crowd', utan riktar sig till oss som jobbar i GitHub, vill ha full kontroll, och bygger seriösa pipelines med versionshantering, CI/CD och testbarhet."

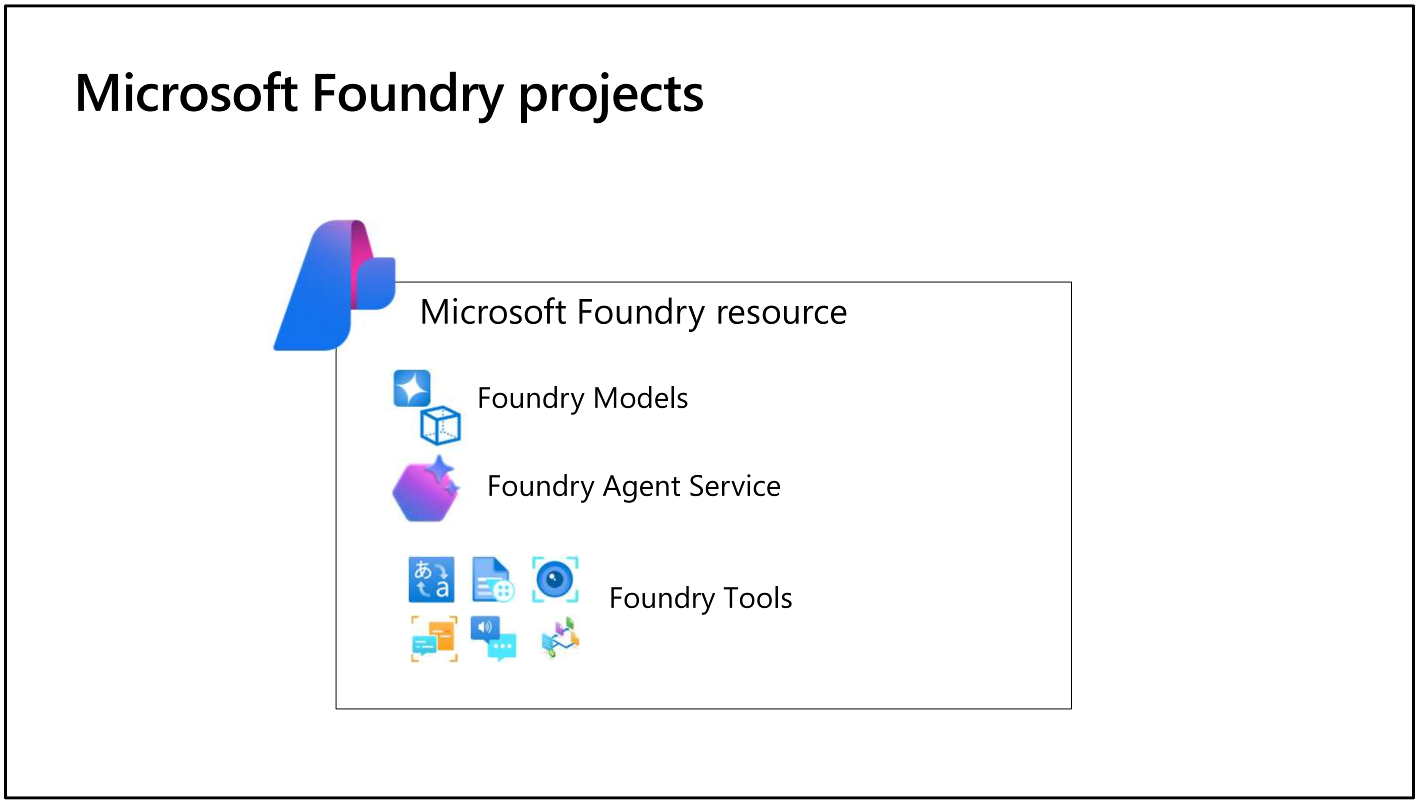
Sedan skulle utvecklaren kanske fortsätta med några konkreta observationer:
"Det som sticker ut är integrationen med Visual Studio, Copilot Studio och GitHub Actions – det gör att jag kan bygga, testa och deploya agents och modeller utan att lämna mitt flöde."
"Jag gillar att jag kan jobba med egna modeller eller använda OpenAI:s genom Azure, utan att

behöva kompromissa med governance eller säkerhetskrav."
"Det känns som Microsoft försöker skapa en slags 'AWS SageMaker för generativ AI', men med tydligare fokus på ansvarstagande AI och enterprise readiness."

Men de skulle kanske också lägga in en brasklapp:
"Det är inte helt moget ännu – verktyg som Copilot Studio är fortfarande ganska GUI-drivna. Men Foundry är ett tydligt tecken på att Microsoft vill bli den naturliga hemvisten för proffsbyggd AI i molnet."

You can start by explaining that **Microsoft Foundry** is a platform that empowers developers to innovate with AI and shape the future. It provides a collaborative environment with enterprise-grade security, where you can explore, build, test, and deploy AI solutions and machine learning models.

Microsoft Foundry portal provides a web-based development portal for professional software developers.



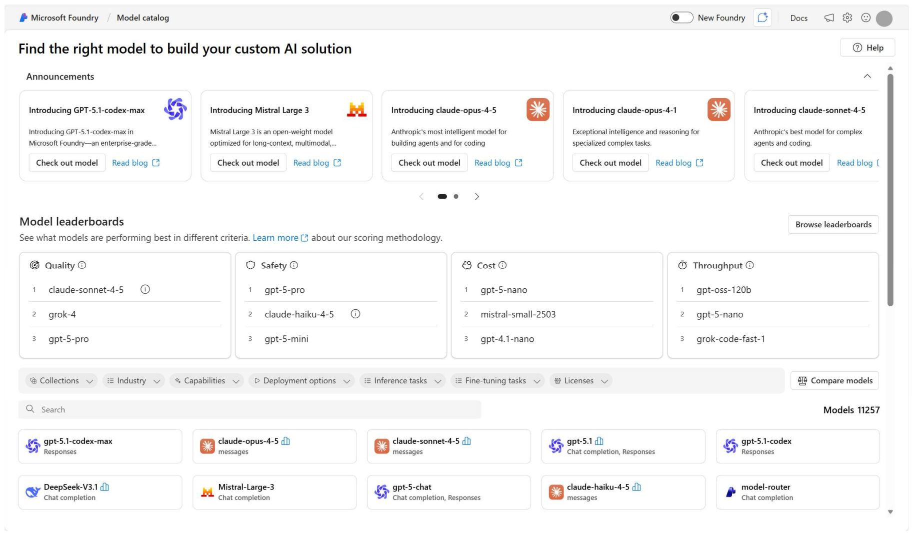
Microsoft Foundry projects

Foundry projects are based on Foundry resources in Microsoft Azure. They provide resources for creating and managing generative AI apps and agents, including a Foundry model catalog with generative AI models from a wide range of vendors, like OpenAI, Microsoft, and others. The Foundry Agent service provides a framework for building agentic solutions in a Foundry project. Foundry projects also include built-in Foundry Tools capabilities for language, vision, and information extraction.

(We'll discuss models and agents in the rest of this section. AI tools are covered later in the course)

Foundry models

- Find the model that best suits your needs in the Foundry model catalog:
  - OpenAI (e.g. GPT 5)
  - Microsoft (e.g. Phi 4)
  - Popular third-party models
- Deploy the model(s) you want to use in your apps



Find the right model to build your custom AI solution

Announcements

Introducing GPT-5.1-codex-max: Introducing GPT-5.1-codex-max in Microsoft Foundry—an enterprise-grade...

Introducing Mistral Large 3: Mistral Large 3 is an open-weight model optimized for long-context, multimodal,...

Introducing claude-opus-4-5: Anthropic's most intelligent model for building agents and for coding

Introducing claude-opus-4-1: Exceptional intelligence and reasoning for specialized complex tasks.

Introducing claude-sonnet-4-5: Anthropic's best model for complex agents and coding.

Model leaderboards

See what models are performing best in different criteria. Learn more about our scoring methodology.

	Quality	Safety	Cost	Throughput
1	claude-sonnet-4-5	gpt-5-pro	gpt-5-nano	gpt-oss-120b
2	grok-4	claude-haiku-4-5	mistral-small-2503	gpt-5-nano
3	gpt-5-pro	gpt-5-mini	gpt-4.1-nano	grok-code-fast-1

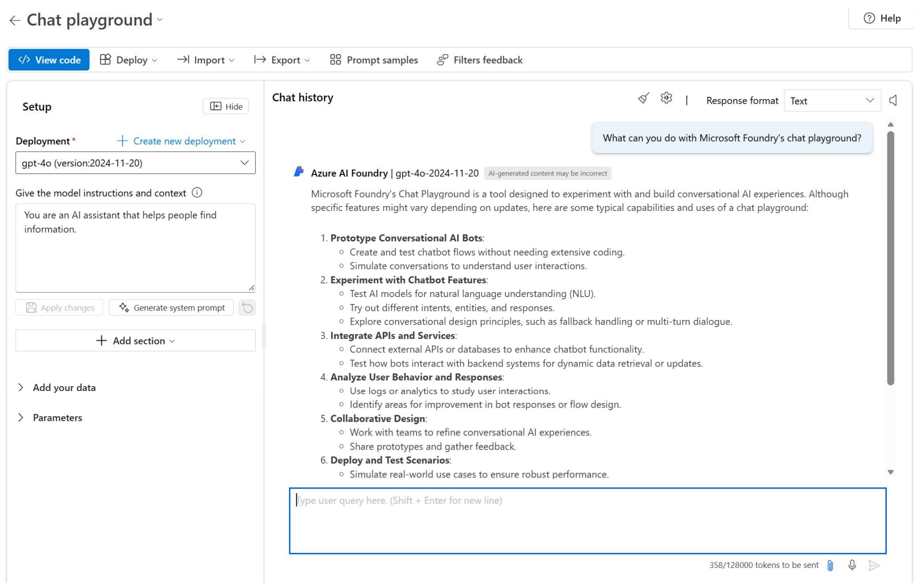
Models 11257

gpt-5.1-codex-max	Responses
claude-opus-4-5	messages
claude-sonnet-4-5	messages
gpt-5.1	Chat completion, Responses
gpt-5.1-codex	Responses
DeepSeek-V3.1	Chat completion
Mistral-Large-3	Chat completion
gpt-5-chat	Chat completion, Responses
claude-haiku-4-5	messages
model-router	Chat completion

The Foundry model catalog provides access to a huge collection of generative AI models from OpenAI, Microsoft, and other vendors. You can use the information and metrics provided for each model to choose the ones that best suit your needs and deploy them in your Foundry project so that developers can use them to build generative AI apps.

Chat playground

- Test Foundry model deployments in the *Chat playground*
- Experiment with questions, instructions, your own data, and parameters
- Deploy your customization as a web app



gpt-4o (version:2024-11-20)

Give the model instructions and context

You are an AI assistant that helps people find information.

What can you do with Microsoft Foundry's chat playground?

Azure AI Foundry | gpt-4o-2024-11-20 AI-generated content may be incorrect

Microsoft Foundry's Chat Playground is a tool designed to experiment with and build conversational AI experiences. Although specific features might vary depending on updates, here are some typical capabilities and uses of a chat playground:

1. **Prototype Conversational AI Bots**:
   - Create and test chatbot flows without needing extensive coding.
   - Simulate conversations to understand user interactions.
2. **Experiment with Chatbot Features**:
   - Test AI models for natural language understanding (NLU).
   - Try out different intents, entities, and responses.
   - Explore conversational design principles, such as fallback handling or multi-turn dialogue.
3. **Integrate APIs and Services**:
   - Connect external APIs or databases to enhance chatbot functionality.
   - Test how bots interact with backend systems for dynamic data retrieval or updates.
4. **Analyze User Behavior and Responses**:
   - Use logs or analytics to study user interactions.
   - Identify areas for improvement in bot responses or flow design.
5. **Collaborative Design**:
   - Work with teams to refine conversational AI experiences.
   - Share prototypes and gather feedback.
6. **Deploy and Test Scenarios**:
   - Simulate real-world use cases to ensure robust performance.

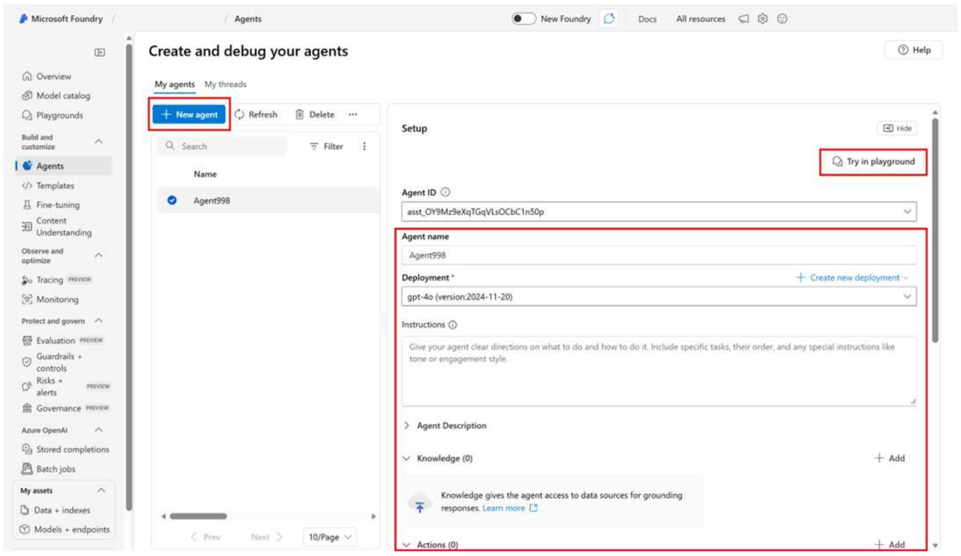
Type user query here. (Shift + Enter for new line)

358/128000 tokens to be sent

Once you have a Foundry Model deployed, you can experiment further with it in the chat playground. In the playground, you can add further customization to how the model behaves in the setup section. Once you are getting the results you are looking for in the chat, you can deploy your specific setup as a web application.

Foundry Agents

- Create agents in Foundry specifying:
  - Agent name
  - Model deployment
  - Instructions
  - Knowledge tools
  - Action tools
  - Connected agents
- Test agents in the *Agents playground*



Agent ID: asst_OY9Mz9eXqTGqVLsOCbC1n50p

Agent name: Agent998

Deployment: gpt-4o (version:2024-11-20)

Foundry includes Foundry Agents service, which provides a platform for creating Generative AI powered AI agents.

When you create an agent, you can specify:

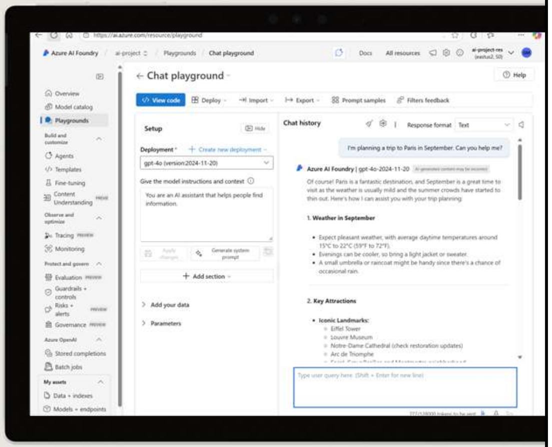
- A meaningful name for the agent
- The Generative AI model deployment the agent should use
- Knowledge tools that provide the agent with access to data sources
- Action tools that provide the agent with the ability to perform programmatic tasks
- Connected agents that enable multi-agent solutions in which each agent performs a particular role.

Exercise – If time permits

Explore generative AI in Microsoft Foundry

In this exercise, you'll use Foundry models to explore generative AI capabilities.

Start the exercise at:
https://go.microsoft.com/fwlink/?linkid=2249955



Deployment: gpt-4o (version:2024-11-20)

You are an AI assistant that helps people find information.

I'm planning a trip to Paris in September. Can you help me?

Azure AI Foundry | gpt-4o-2024-11-20

Of course! Paris is a fantastic destination, and September is a great time to visit as the weather is usually mild and the summer crowds have started to thin out. Here's how I can assist you with your trip planning:

1. Weather in September

- Expect pleasant weather, with average daytime temperatures around 15°C to 22°C (59°F to 72°F).
- Evenings can be cooler, so bring a light jacket or sweater.
- A small umbrella or raincoat might be handy since there's a chance of occasional rain.

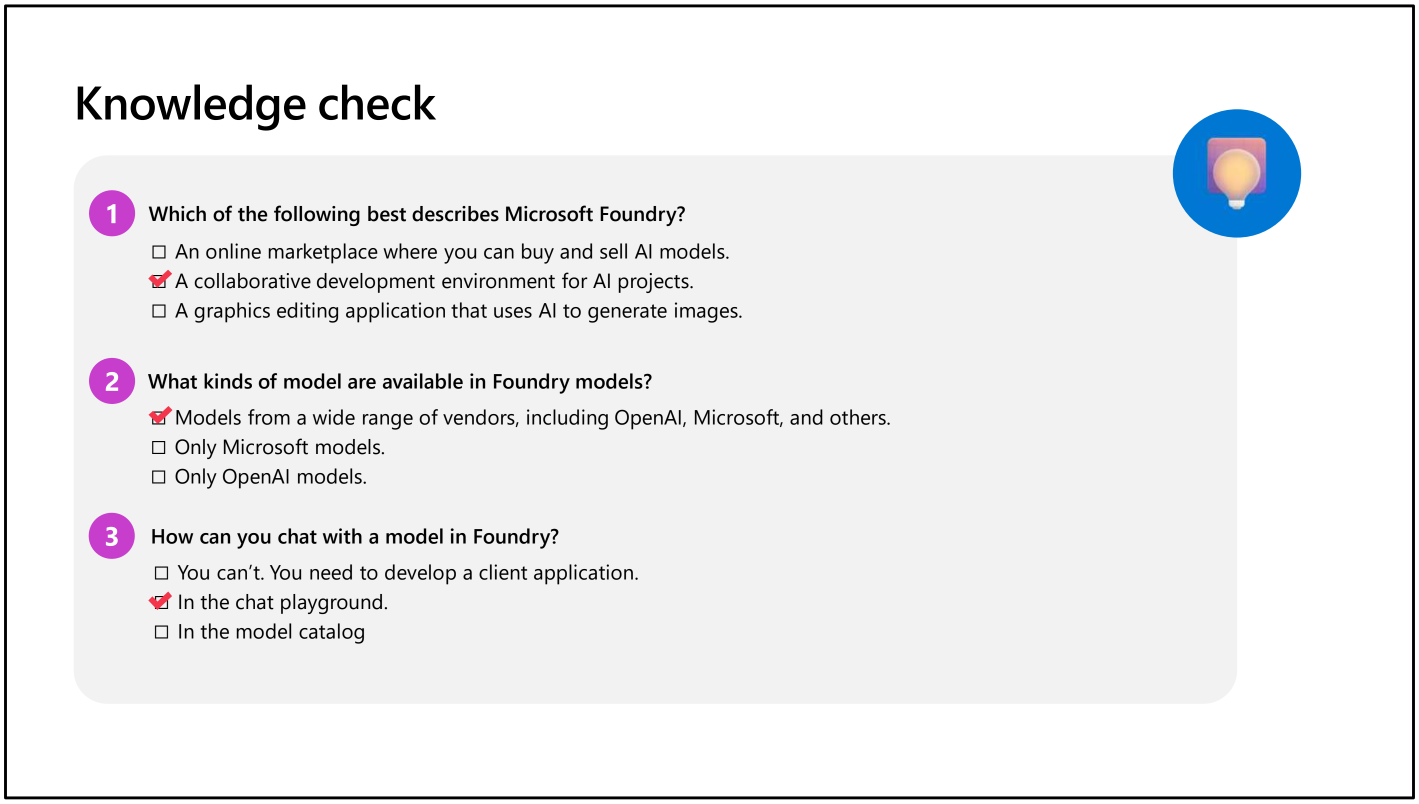
2. Key Attractions

- Iconic Landmarks:
  - Eiffel Tower
  - Louvre Museum
  - Notre-Dame Cathedral (check restoration updates)
  - Arc de Triomphe

You require an Azure subscription to perform this exercise – which may be provided by an Authorized Lab Hoster.

The exercise can take a substantial amount of time. Our recommendation in one-day deliveries is for the instructor to demonstrate the core tasks in this exercise; completing the exercise ahead of time (https://go.microsoft.com/fwlink/?linkid=2249955) so you have a project containing a deployed model to demonstrate.

Om problem, kör oai.azure.com



Knowledge check

1 Which of the following best describes Microsoft Foundry?

- ☐ An online marketplace where you can buy and sell AI models.
- [✓ A collaborative development environment for AI projects.]
- ☐ A graphics editing application that uses AI to generate images.

2 What kinds of model are available in Foundry models?

- [✓ Models from a wide range of vendors, including OpenAI, Microsoft, and others.]
- ☐ Only Microsoft models.
- ☐ Only OpenAI models.

3 How can you chat with a model in Foundry?

- ☐ You can't. You need to develop a client application.
- [✓ In the chat playground.]
- ☐ In the model catalog

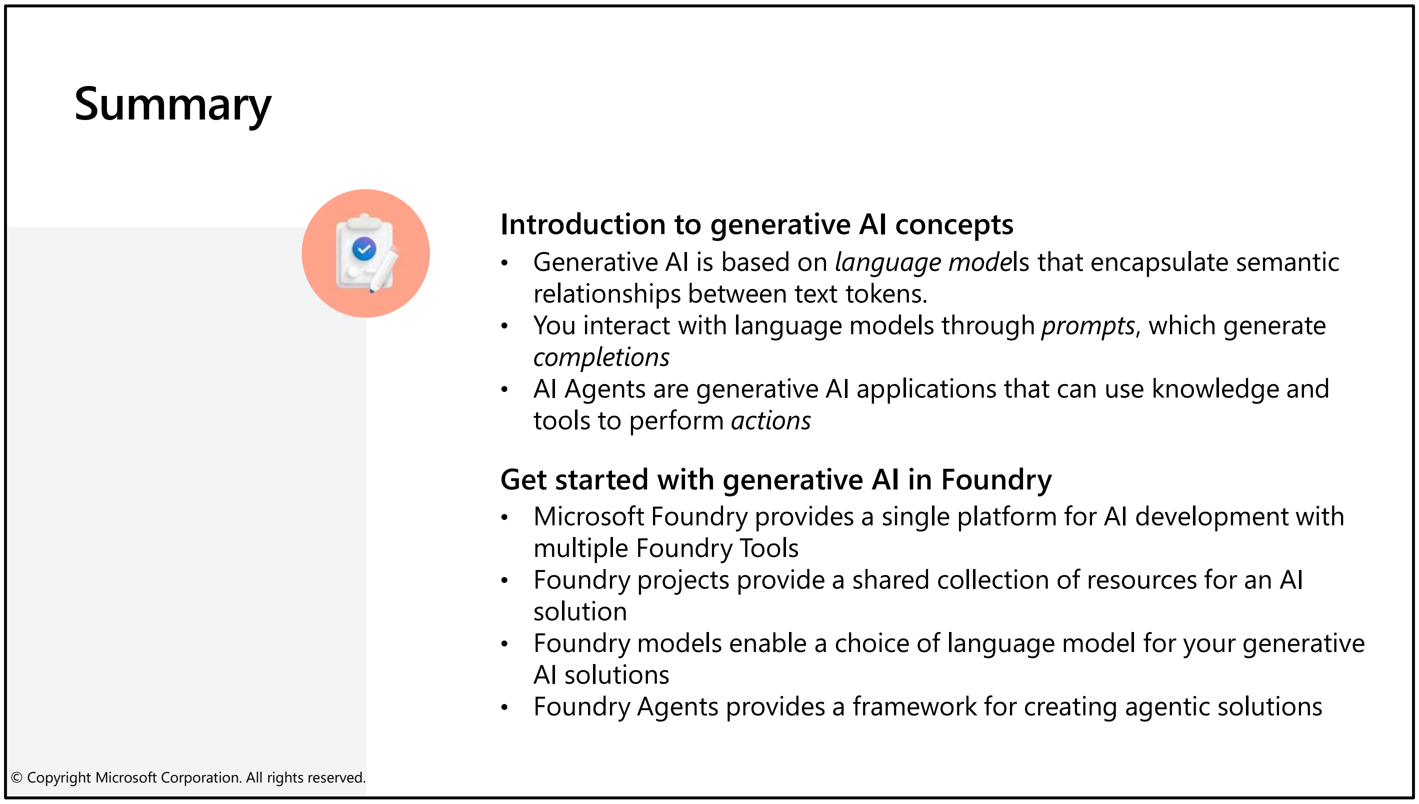
Här är korta svar med **max 10 ord per motivering**:

1. Microsoft Foundry
❌ **Fel:** Är inte en marknadsplats för handel.
✅ **Rätt:** Samarbetsmiljö för att bygga och hantera AI-lösningar.
❌ **Fel:** Inte ett bildredigeringsverktyg.

2. Modeller i Foundry
✅ **Rätt:** Samlar modeller från flera leverantörer.
❌ **Fel:** Begränsat inte till endast Microsoft.
❌ **Fel:** Innehåller fler än OpenAI-modeller.

3. Chatta med modell i Foundry
❌ **Fel:** Kräver inte egen klientutveckling.
✅ **Rätt:** Direkt interaktion via inbyggd chattmiljö.
❌ **Fel:** Katalogen är för bläddring, inte chatt.

Allow students a few minutes to think about the questions, then reveal the correct answers.



Summary

Introduction to generative AI concepts

- Generative AI is based on *language model*s that encapsulate semantic relationships between text tokens.
- You interact with language models through *prompts*, which generate *completions*
- AI Agents are generative AI applications that can use knowledge and tools to perform *actions*

Get started with generative AI in Foundry

- Microsoft Foundry provides a single platform for AI development with multiple Foundry Tools
- Foundry projects provide a shared collection of resources for an AI solution
- Foundry models enable a choice of language model for your generative AI solutions
- Foundry Agents provides a framework for creating agentic solutions