[20]

Intro to

Large Language Models

(LLMs)

* [**https://www.codecademy.com/enrolled/courses/intro-to-llms**](https://www.codecademy.com/enrolled/courses/intro-to-llms)
* [**https://www.codecademy.com/courses/intro-to-llms/lessons/language-models-and-text-generation/exercises/autoregressive-language-models**](https://www.codecademy.com/courses/intro-to-llms/lessons/language-models-and-text-generation/exercises/autoregressive-language-models)

**Introduction to Large Language Models (LLMs)**

### **Chatbots, Language Models and the Birth of AI**

This lesson is an introduction to Large Language Models (LLMs) and text-based Generative

[Artificial Intelligence](https://www.codecademy.com/resources/docs/general/artificial-intelligence)

Preview: Docs Simulates human intelligence in computers, enabling learning, reasoning, and problem-solving to provide solutions across various tasks.

(AI). These technologies power many sophisticated text-based AI applications we see today, such as ChatGPT, released by OpenAI in late 2022. Before we jump into how today’s models function, we’ll start back in 1966 to understand the history of chatbots, text generation, and the Turing test.

###### **The ELIZA Effect**

The first digital chatbot, ELIZA, debuted on the Massachusetts Institute of Technology (MIT) campus in 1966. It was programmed using

[binary](https://www.codecademy.com/resources/docs/general/binary)

Preview: Docs Loading link description

code by computer scientist Joseph Weizenbaum. The program generated text by following a few simple rules. These rules were based on a pre-determined “script” that allowed it to assume a specific persona.

For instance, the version of ELIZA that impressed staff and students alike at MIT was known as “DOCTOR”, a kind of barebones psychotherapist who would mirror back statements typed in by the user of the program. So if someone typed in “I’m feeling X,” DOCTOR would respond with something like “What is making you feel X?”

ELIZA was very successful in convincing people that they were interacting with a human and not a computer! So much so that there’s even a psychological effect named after the program, *the ELIZA effect*. It refers to the tendency to unconsciously attribute “humanness” to (or *anthropomorphize*) computer behaviors.

###### **The Turing Test**

Chatbots have come a long way since the days of ELIZA. From a computer science perspective, ELIZA and ChatGPT can both be said to pass what is known as the “Turing test”.

In 1950, computer scientist

[Alan Turing](https://www.codecademy.com/resources/docs/general/historical-technical-figures/alan-turing)

Preview: Docs Loading link description

wrote a landmark paper titled [“Computing Machinery and Intelligence”](https://academic.oup.com/mind/article/LIX/236/433/986238), in which he investigated the question:

What does it mean for machines to think?

Copy to Clipboard

He concluded that this question was fundamentally meaningless and said that the better question to ask would be:

Can a machine successfully imitate human behavior so that it might dupe a human?

Copy to Clipboard

To answer this question, he devised a game. The “imitation game”, as he called it, has a human evaluator interacting with a machine and another human through a text-only channel. A machine is said to do well at the imitation game if the human evaluator is convinced that they were interacting with another human and not a machine. The imitation game is referred to today as the Turing test.

###### **“AI”: what is it?**

The Turing test has been a cornerstone in computer science to assess the intelligence of a machine and a lot of the detail lies in the design of the test itself. Six years after Turing’s paper was published, a group of scientists put together a proposal for a Summer Research Project on Artificial Intelligence (AI) at [Dartmouth](https://home.dartmouth.edu/about/artificial-intelligence-ai-coined-dartmouth), often thought of historically as the moment that birthed the field of AI.

Instead of trying to define what AI is or is not, they sought to explain the fundamental assumption behind the pursuit of AI:

“The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.”

Copy to Clipboard

Specifically, they were interested in how language might be useful in this pursuit :

“An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves."

Copy to Clipboard

From ELIZA to ChatGPT, the abilities demonstrated by language-based technologies have indeed increased in sophistication, nuance, and complexity. What has made this possible was the move away from “rule-based” chatbots and towards language models that rely on neural networks to capture statistical regularities within language. Let’s dive into how these models work!



### **Detecting Patterns in Text**

The quest to mathematically model language dates back over a hundred years. In 1913, mathematician A.A. Markov analyzed letters in a Russian novel and found that some surprising patterns could be captured by mathematics. Computer scientist and mathematician Claude Shannon followed up on this analysis a few decades later with a broad question about language patterns: *Can we predict the next letter given a series of letters?*

###### **Next Letter Prediction**

For example, consider the XKCD comic shown to the right, with the three-letter sequence “eru-. “Only a few English words begin with this combination of letters, and they are rare. The words “erudite,” “erupt,” and the less often used “eructate” are a few possible candidates, for instance.

The context of the sentence also narrows down the possibilities for which word can occur next and still make sense. The appearance of “volcano” implies that some variation of the verb “erupt” might be an obvious choice. The verb tense (in this case, present continuous) gives the final clue that the word “erupt” will appear in the form “erupting”.

###### **Natural Language Processing (NLP)**

The explanation above relied on familiarity with the English language. Specifically, we relied on the patterns in language we’ve picked up from reading and speaking to inform our thinking as we solved this. Can we train an

[algorithm](https://www.codecademy.com/resources/docs/general/algorithm)

Preview: Docs Loading link description

to do the same, possibly faster and on many sentences simultaneously? The answer is an emphatic yes, and this is the primary goal of the field of “Natural Language Processing” (NLP)! To automate this process using a computer, we must devise a way to turn this text into math.

Many initial steps in any NLP task involve standardizing things (i.e., treating language more like numbers, which lend themselves more easily to math) so that computers can analyze text.

Copy to Clipboard

###### **From Letters to Words to Tokens**

What is the most straightforward language unit to which text can be broken down? A letter! A single-letter unit is known as a “unigram”. The sentence “Oh my god, the volcano is eru-“can be broken down into a series of unigrams as follows: {o, h, \_, m, y, \_, g, o, d, \_, t, h, e, \_, v, o, l, c, a, n, o, \_, i, s, \_, e, r, u} where the dashes (\_) represent blank spaces. We could also break it up into two letter units, known as bigrams: where the units would look like {oh, h\_, \_m, my, etc,} or three letter units, i.e. trigrams and ultimately n-letter units, known as n-grams.

The best choice of the unit length depends on the exact problem we’re trying to solve, and there are many best practices to follow around this. Formalizing language this way allows us to build a language model on a computer to predict the next best unigram, bigram or n-gram.

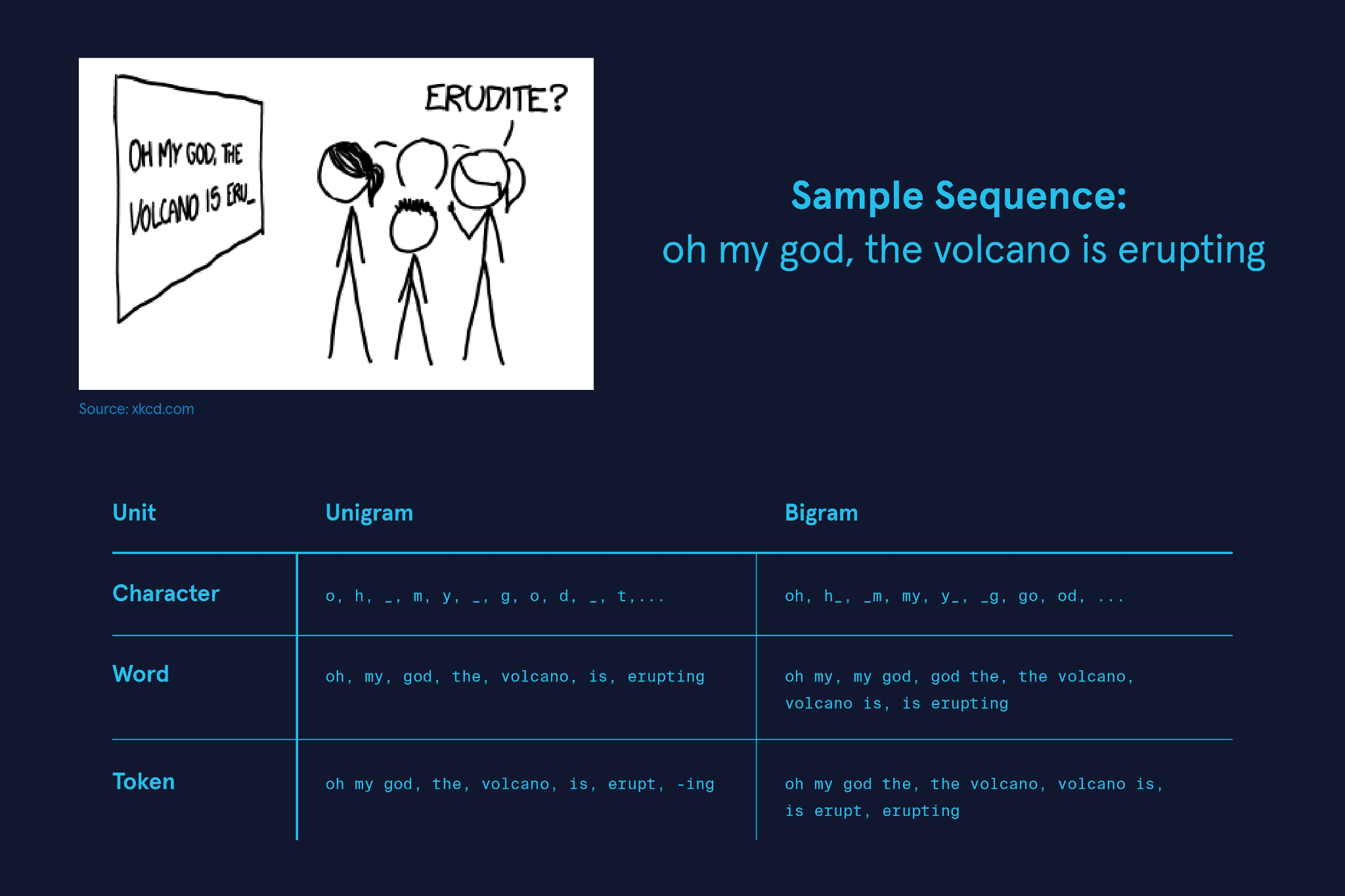
Can we do this with only letters? Words are a more natural language unit; we could use them as our smallest unit. We can identify each word unit by the appearance of the spaces between them. The [Google n-gram viewer](https://books.google.com/ngrams/) uses words as language units to give us statistical insights into the appearance of words in Google Books.

**We could also use groups of words or subwords (like -ing, -ed, etc.), often referred to as tokens in the context of language models.**

The table to the right shows what this might look like for the sentence in the comic.

**Instructions**

Search the phrase “artificial intelligence” in Google Books’ [n-gram viewer](https://books.google.com/ngrams/) to view the history of the appearance of the phrase in the Google Books data base!



**Introduction to Large Language Models (LLMs)**

### **Autoregressive Language Models**

We’ve seen in the previous exercise that the first step in any NLP task is to figure out how to turn text into numbers so we can do math with it. Before getting to the math, let’s consider which tasks are well-suited for NLP. Some well-known language-related tasks that NLP algorithms are involved in are

* translation (like in Google Translate)
* completing letter sequences (autocomplete, for instance)
* question-answer (customer service chatbots, for example)
* sentiment analysis (used in content filters)

While there are many methodologies in NLP to do these tasks, the quest to build an algorithmic system that can do all of these tasks and more excites NLP researchers the most. Such a model is referred to as a **language model**. Since language is often the vehicle of human thought, the hope is that in building larger language models, an algorithmic system might exhibit a kind of generalized intelligence that’s worthy of comparison to humans. (*This is a big and interesting question on its own.*)

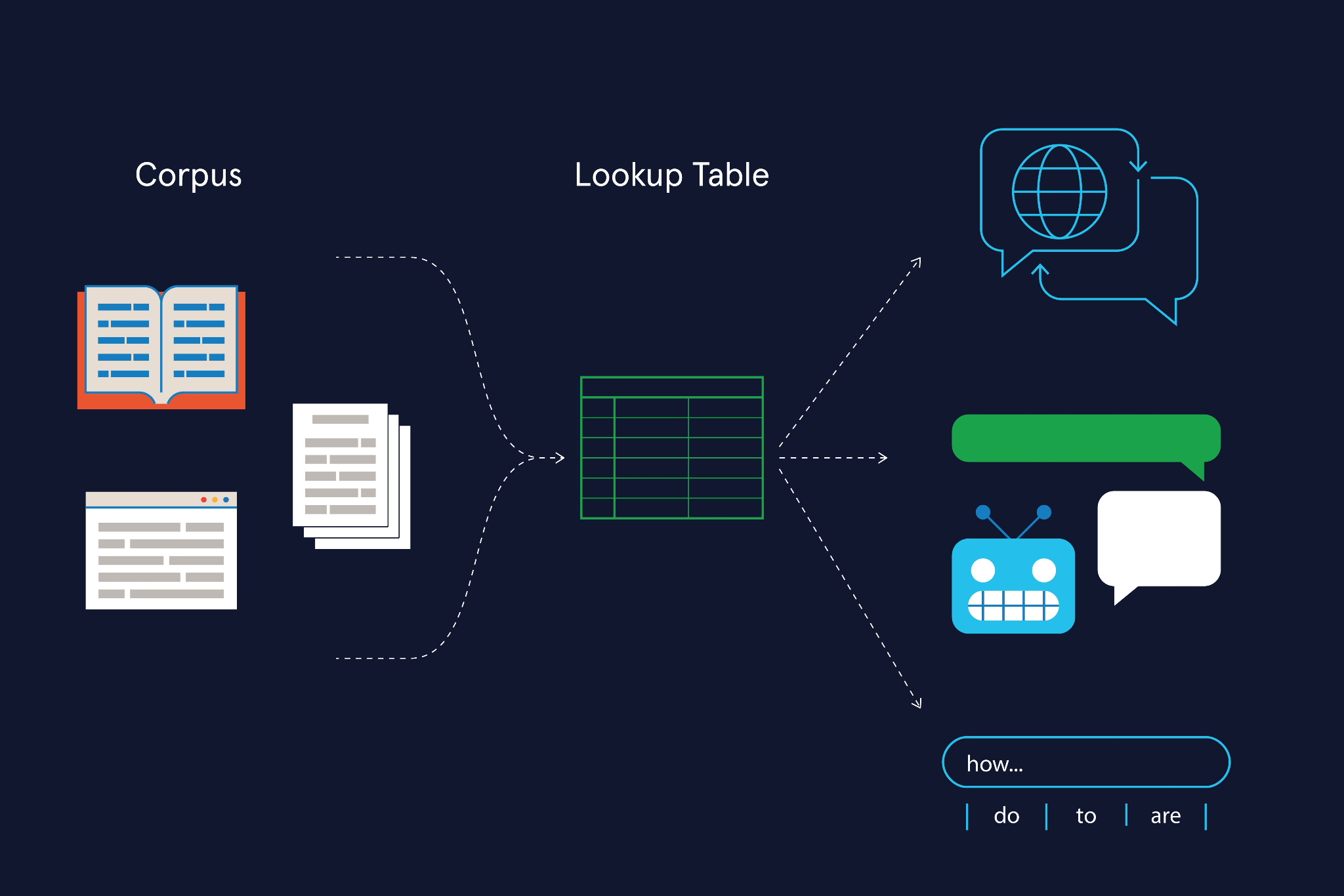
Language models can get large and complex, but, the modelling process begins with a very simple question: Given some text data to work with, how can one build an

[algorithm](https://www.codecademy.com/resources/docs/general/algorithm)

Preview: Docs Loading link description

to predict the next best thing to say? As it turns out, a model that answers this question well is capable of doing many other language-related tasks! Text data can come in many forms, such as a book, a collection of articles, a set of websites, etc. It is often called as a [“corpus”](https://en.wikipedia.org/wiki/Text_corpus) in NLP to indicate that it is likely gathered from many sources and digitally stored.

The word **autoregressive** refers to statistical models that predict future values based on past values. It is used very often in the context of language models and it means that we’re using previously occurring words (or word sequences) to predict the next best thing to say. The simplest way to build a language model from a text corpus is to build a giant lookup table. This table contains all possible word combinations appearing in the text and their frequency of appearance so we can easily look up how likely it is that a given sentence occurs in the text. Such a model is often referred to as a **count-based language model**.



**Introduction to Large Language Models (LLMs)**

### **Count-based Autoregressive Language Models**

Let’s examine this with an example. Consider a text corpus from which we want to predict how likely it is that the sentence “What do I say next?” appears. We would build a giant lookup table like the one shown on the right. The questions we would try to answer would be in the following order:

- How likely is it that the word `what` appears?

- How likely is it that the word `do` appears after the word `what`?

- How likely is it that the word `I` appears after the sequence `what do`?

Copy to Clipboard

… and so on!

The mathematical way of looking at this would be through *probabilities*. In the figure shown to the right, the symbol P(do | what) refers to what is known as a *conditional probability*:

`P(do | what)` is the probability that the word `do` appears given that the word `what` has already appeared.

Copy to Clipboard

How can we calculate this? From the lookup table we see four possibilities with varying frequencies of occurrence: “what do” appears 40 times in the text; “what am”, 16 times; “what should”, 16 times and “what have”, 8 times. So we can estimate the probability, P (do | what) (to be read out aloud as *P of “do” given “what”*) thus:

*P*(do∣what)=4040+16+16+8=4080=0.5

Now, the probability that the sequence “what do” appears is not just P(do | what) but rather:

P(what do)=P(what)∗P(do∣what)

, i.e., the probability that the word “what” appears to begin with, multiplied by the probability that it is followed by “do”.

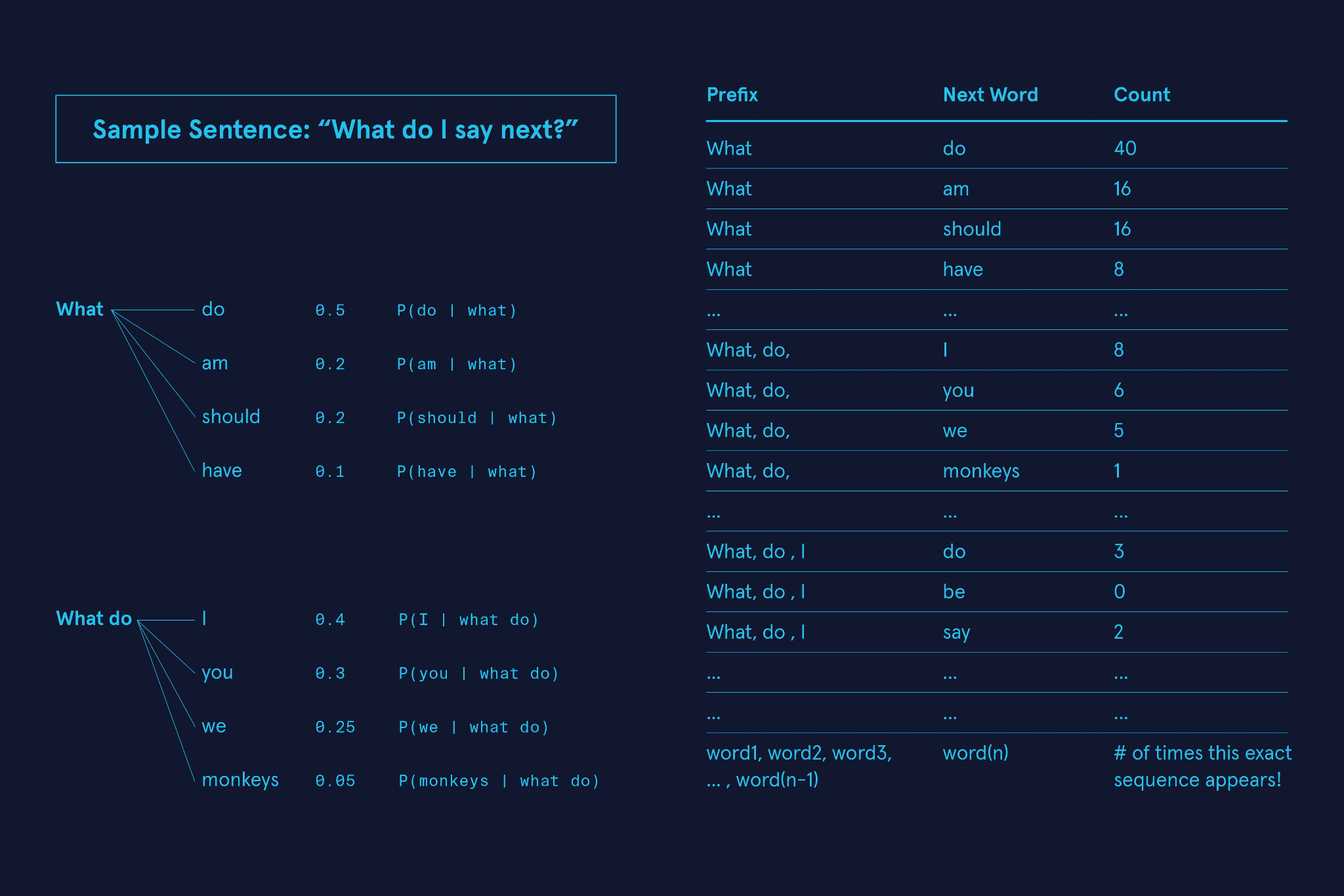
Expanding this further, the probability that the sentence “What do I say next?” appears in this corpus of text would be:

P(sentence)=P(what)∗P(do∣what)∗P(I∣what do)∗P(say∣what do I)∗P(next∣what do I say)

*Note*: A typical corpus usually contains many more possibilities, but we’ve kept it short and sweet for the sake of this exercise!

**Instructions**

Can you examine the counts of the next word to see if the probabilities match up? For instance, does the probability of the occurrence of “monkeys” after “what do” match the count tables?  
You can use a similar approach as what we did in the exercise. We see four possibilities in our corpus to follow “what do”, and they are “I”, “you”, “we”, and “monkeys”. Examine the frequency of occurrence of “monkeys” after “what do” and divide it by the sum of counts of all 4 possibilities to get P(monkeys | what do). Based on this, what would P(what do monkeys) be?



**Introduction to Large Language Models (LLMs)**

### **Generalization: From Count-based to Neural Language Models**

What happens if a sentence never occurs in the text that a count-based language model is trained on? Consider a sentence that is highly unlikely to occur in known text like the following:

A lion is chasing a llama.

Copy to Clipboard

It’s rather unlikely for these two animals to interact in real life, since they exist on different continents, and even less likely that there is a piece of writing describing such an event. It is *conceivable* however, that a predator like a lion *could hypothetically* chase a domesticated animal like a llama. But a count-based language model will assign a zero probability if this *exact* sequence of words does not appear in the text it has been exposed to!

The inability to work with possibilities that do not appear in training data is because count-based models are unable to *generalize*. Generalization is the ability of a model to adapt to new and unseen data. In the context of language models generating text, it means that they can produce text that doesn’t exist in its training data.

To get a language model to generalize, we would need to **move beyond counting words** and instead, find a way to link words to their meaning and context. This type of model uses **semantics** (i.e., what words mean) rather than counting (which words exist in text). For instance, a semantic language model might map “lions” to the idea of predators and “llamas” to cattle/prey/domesticated animals. This allows it to use the connection that predators chase prey and even assign a non-zero probability for lions chasing llamas. This is often known as a “semantic representation”. The mathematical objects that allow language models to generalize this way are known as **word embeddings**.

Word embeddings allow us to turn each word into a series of numbers, also known as a word vector. These vectors exist in an abstract space where words are semantically linked, i.e., they’re linked by the meaning and context in which they appear. This makes for a more sophisticated model. One example would be the way it treats homonyms, i.e., two words that have the same spelling but different meanings. For instance, the word “lead” is different when paired with the word “President” than it is when paired with “pipe”. A language model using word embeddings would be able to distinguish between homonyms.

The most popular and efficient way to build such embeddings from text is using **neural networks** and language models built using neural networks are referred to as **neural language models**.

*Note 1*: “Generalizing” and “generative” are similar sounding words used in the context of language models that mean different things:

* “Generative” refers to the *ability to generate* data and this might be text/image/audio/video depending on the kind of model being built.
* “Generalizability” refers to the *ability to generalize* , i.e., the ability to adapt to or produce unseen data.

*Note 2*:

* The phrase “non-zero probability” means “it is possible that…”
* The phrase “zero probability” means “it is impossible that…”

**Instructions**

This example, illustration and explanation is inspired by AI researcher (and New York University professor) **Prof. Kyunghun Cho**‘s excellent talk on [“*A slight-less-magical perspective into autoregressive language modeling: count, compress and prune*“](https://www.youtube.com/watch?v=OZtRlORfd28&t=1414s). You can check out more talks and amazing material on language models and text generation on his webpage [here](https://kyunghyuncho.me/talks/)!  
Now, if we perform the kind of probabilistic calculations that we did in the previous exercise for the sentence “The lion is chasing a llama”, it might look something like the image shown to the right. Do the probabilities shown in the image seem reasonable to you?



**Introduction to Large Language Models (LLMs)**

### **Compression: Solving the Curse of Dimensionality**

###### **Count tables can get huge!**

The other issue that count-based language models run into is referred to as the curse of dimensionality. This means that as the corpus of text gets bigger, so does the count table we would need to construct. To store and work with giant count tables requires enormous amounts of computing memory and speed. Suppose we had a document with a hundred words, and we wanted to calculate the probability of every five-word sentence. The number of combinations we have would be 100^5 = 10 billion counts! We can see how this problem quickly scales with the size of a text corpus.

###### **Compressing Text by Approximation**

Neural language models help solve the curse of dimensionality by *compressing* the text into smaller number of parameters, often referred to as the “weights” of a model. It does so by trying to learn *an approximation* of the count table instead of the whole count table.

One way to think about this is to imagine reducing a high resolution photo to a lower resolution image. This reduced image will be easier to store and access than the higher resolution one and the extent to which it retains all the important features of the original image depends on the cleverness with which it is compressed! (The science fiction writer Ted Chiang provides a layman’s explanation of this for ChatGPT in his excellent [piece in the New Yorker](https://www.newyorker.com/tech/annals-of-technology/chatgpt-is-a-blurry-jpeg-of-the-web).)

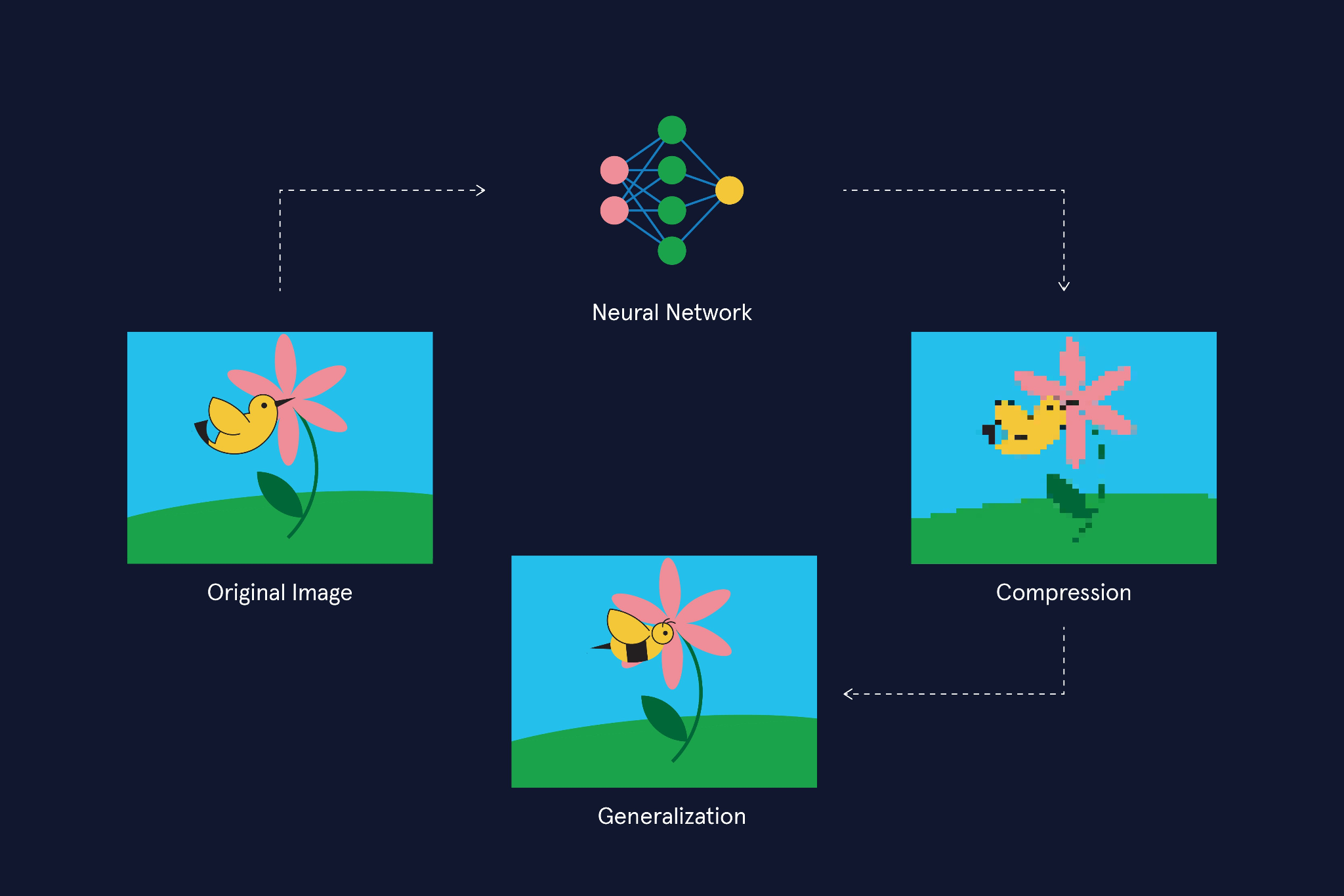
###### **Information Loss**

If you’re wondering “Is there information loss in this process?”, the answer is yes, there is! Neural language models run the risk of sometimes assigning zero probabilities to text that actually exists in the corpus, thus running into the unavoidable issue of information loss. However, it is compression that allows a neural language model to generalize in a manner that produces novel/unseen text! This is because it allows semantic connections to be made between between (lion → predator), (predator → chasing prey) and (llama → prey) to assign a non-zero probability to a sentence like “A lion is chasing a llama”.

To summarize, neural language models:

* can assign zero probabilities to existing text (compression)
* can assign non-zero probabilities to unseen text (generalization)

In other words, compression and generalization are two sides of the same coin! We will examine this deeper in a couple of exercises but before that, let’s look into how neural networks are able to accomplish this feat.



### **Neural Networks and Language Models**

###### **Revisiting ELIZA and Symbolic AI**

In the history of Natural Language Processing (NLP), neural networks have become prominent only fairly recently, even though they were developed around the same time as ELIZA. ELIZA is an example of an attempt at building what is known as “*symbolic AI*“. The field of symbolic AI attempted to build rule-based systems grounded in mathematical logic so as to capture conscious thought processes. ELIZA, for instance, responded to any question by drawing from a long list of explicit rules on how to respond — a semi-scripted conversation with many different endpoints.

###### **Neural Networks and Subsymbolic AI**

Neuroscience developments in the 1950’s suggested that *unconscious processes* play an important role in our brains, informing human perception and cognition, and that this cannot be captured by explicit rule-making. Computer scientists who drew inspiration from this came up with the idea of “*subsymbolic AI*“ which seeks to capture these underlying unconscious processes.

In 1958, the psychologist, Frank Rosenblatt, built the “*perceptron*“ — a simple program meant to simulate the function of an individual information processing unit within the brain, the neuron. The idea was that by layering networks of these perceptrons, a form of intelligent information processing that brains perform would emerge in these **neural networks**.

###### **Deep Learning**

Neural networks went in and out of favor multiple times among AI researchers for nearly fifty years until we arrived at the age of data in the early 2000s. The growth of the internet meant that more data and computing power were available than ever before and neural networks saw a resurgence through the field of **machine learning**.

Machine learning refers to a class of algorithms that learn patterns from vast amounts of data to perform tasks like prediction, inference and generation. Neural networks can be shallow (few layers) or deep (many layers). **Deep learning** is a subset of

[machine learning](https://www.codecademy.com/resources/docs/general/machine-learning)

Preview: Docs Machine learning is a branch of artificial intelligence that enables systems to learn from data and make predictions or decisions without explicit programming.

methods that use many-layered or deep neural networks.

###### **Birth of the Transformer**

There are many different types of neural networks and these networks can be arranged in specific ways, to excel at specific tasks. These different arrangements are often referred to as “neural network architectures”. Convolutional Neural Networks (CNNs) are most commonly used for image data. Recurrent Neural Networks (RNNs) work well with sequential data (like language!) They’re used extensively in NLP tasks like translation and speech recognition since the mid 2000s. The image to the right shows the workflow of a RNN translating a Hindi sentence to English.

In 2017, a specific type of architecture, known as the *transformer* was shown to perform exceptionally well in language-related tasks. Generative Pre-trained Transformers or GPTs are a class of Large Language Models (LLMs) that employ transformers to learn from vast text corpuses to be able to generate coherent “human-sounding” text.

How transformers capture semantic patterns within text is beyond the

[scope](https://www.codecademy.com/resources/docs/general/scope)

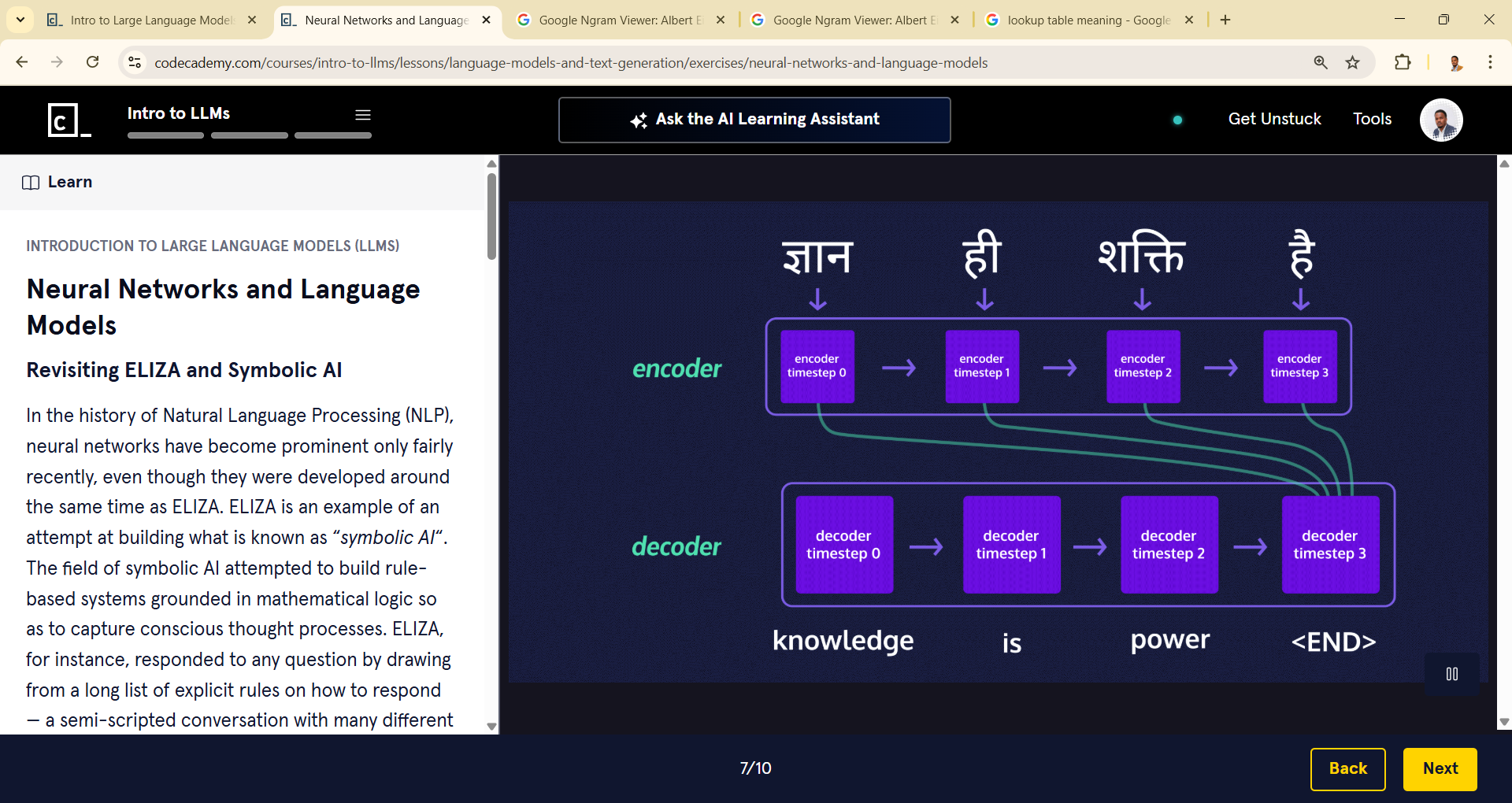
Preview: Docs Loading link description

of this lesson, but we can nonetheless understand many important ideas about how LLMs work, their possibilities and limitations based on what we’ve covered in the lesson so far. Let’s dive into these next!

**Instructions**

1. The language translation task to the right is typically performed by a type of RNN known as a sequence-to-sequence (referred to as seq-to-seq in short) model. And it is exactly as it sounds - it’s designed to take an input sequence and convert it to an output sequence!  
   These models are *built on top of a language model* by adding an encoder and decoder step:
   * The encoder that maps the Hindi text to a mathematical representation, one word at a time.
   * A decoder that maps the mathematical representation to English text.

It’s crucial that the mathematical representation is able to retain the meaning and context of the words as much as possible. Transformer-based language models far outperform previous language models in this regard.



### **LLMs: Caveats and Possibilities**

###### **Large Language Models (LLMs)**

So how are large language models different from small language models? Many of the ideas we’ve covered are applicable to both and let’s recap these:

* Language models need training data in the form of text corpuses and their outputs are dictated (to varying extents) by what is or is not present in their training data.
* Most tasks performed by a language model rely on predicting the next best word (or token!) to say.
* Next word predictions rely on probability distributions generated by the language model. These probability distributions are learned using neural networks.
* Neural networks in a language model convert text data into mathematical representations known as embeddings such that the meaning and context of words gets preserved.
* Neural language models solve the problems of generalization and the curse of dimensionality by compressing their training text into a semantic representation of the text.

Compression allows for the models to generalize, i.e., generate text that *does not exist in training data* but also results in some amount of information loss, i.e., there is information in the training data that might be missing in the compressed model. Now, as the amount of training data for LLMs are increased, while they get bulkier and more computationally expensive to store, some interesting things begin to happen.

###### **Emergence**

Emergence refers to a sudden and sharp increase in model performance in LLMs. It’s an idea that is drawn from the study of complex systems in Physics and physicist Phillip Anderson describes it thus:

"Emergence is when quantitative changes in a system result in qualitative changes in behavior."

Copy to Clipboard

In the context of LLMs like the GPT models, this refers to the occurrence of complex and sophisticated abilities such as providing joke explanations, producing counterfactuals, imitating authors’ styles and writing poetry for example. LLMs exhibit these remarkable qualities even though they were not specifically trained for it, leading people to speculate if they have a more generalized emergent intelligence, often referred to as “Artificial General Intelligence” or AGI.

###### **Hallucinations**

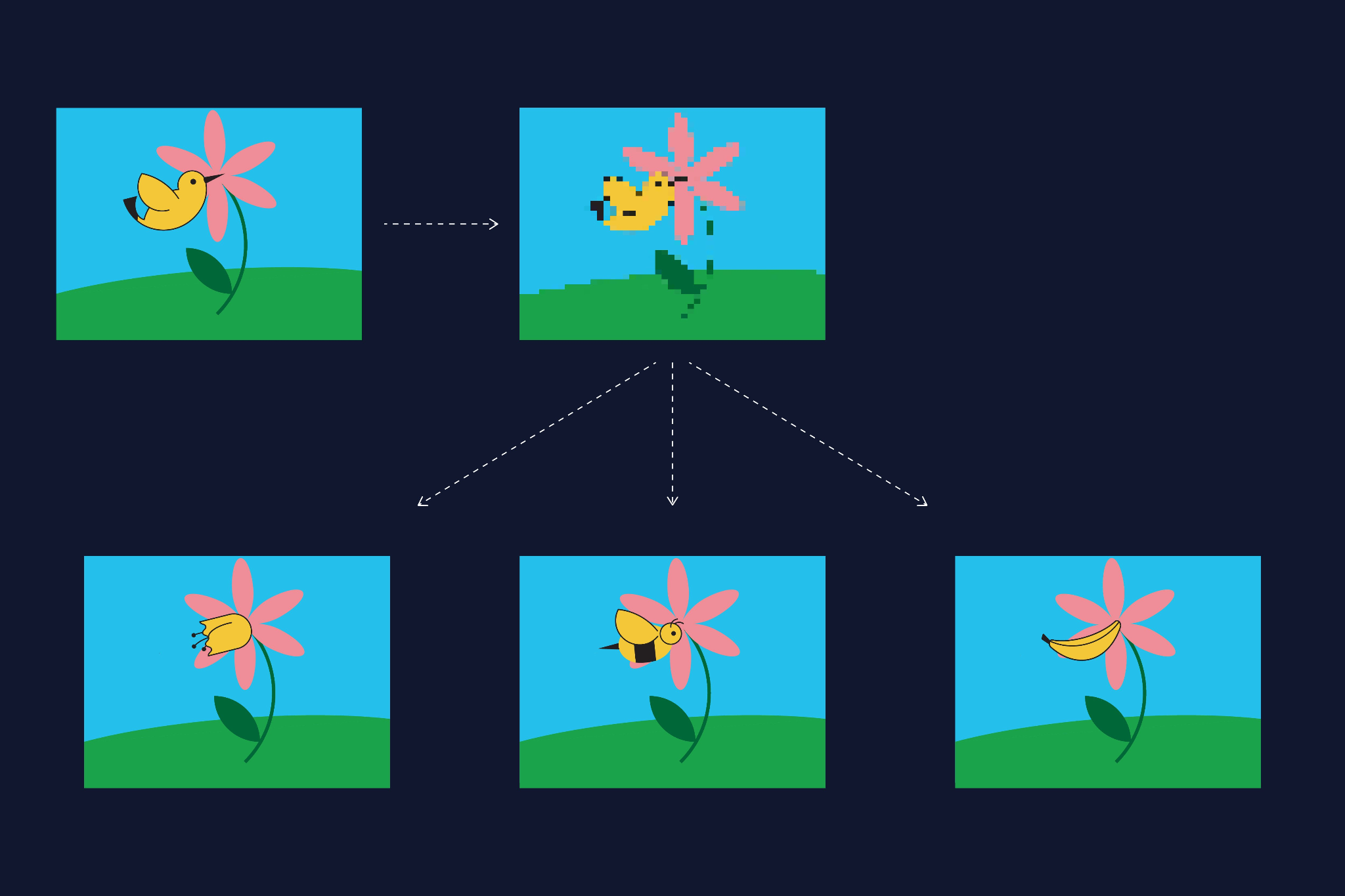
While generalization leads to impressive feats in LLMs, it also presents a seemingly unresolvable issue in them — the question of factual grounding. By its very definition, *unseen text* implies that there is no way for it to have been verified. It hasn’t existed before ergo it has not been verified by a human. The phenomenon where a model produces incorrect or erroneous text is referred to colloquially as “hallucinations”.

###### **LLMs and Factual Grounding**

Language models lack the knowledge of what is grounded in facts and what isn’t, and they rely on the veracity of the text they’re trained on. If they don’t have access to a certain piece of information on a topic, generalization allows them to make up information based on the most plausible thing to say. Oftentimes, this might be the coherent and reasonable thing to say on that topic but some times it might just be absurd or erroneous. (And either way it could be untrue!)

Compression further exacerbates this issue by causing information loss in the first place but even if there were no information loss, the pre-trained model is still static and unable to incorporate new facts, keep up with changing information and most of all *unable to know what it doesn’t know*.

While the latest models such as GPT-4 and PaLM use human feedback through to correct incorrect model outputs and minimize hallucinations, there is no way of completely eliminating them based on how they’re built!



### 

### **LLM Parameters: Temperature**

<https://www.codecademy.com/courses/intro-to-llms/lessons/language-models-and-text-generation/exercises/parameters-in-ll-ms-ap-is-temperature>

Now that we’ve learned about the many facets of text generation using language models, we can look at how we can tweak them to get better results. Whether one is working with LLM’s via chat or

[API](https://www.codecademy.com/resources/docs/general/api)

Preview: Docs Loading link description

, there are parameters in the models that allow us to play around with the probabilities involved in the models. One of the most used parameters is **temperature**.

Temperature is a

[parameter](https://www.codecademy.com/resources/docs/general/parameter)

Preview: Docs Loading link description

that controls the sharpness of the probability distribution that the model generates. This means that it increases the probability of the higher-probability outcomes and decreases the probability of the lower-probability outcomes, i.e., the distribution becomes narrower, as shown to the right.

Let’s understand this with an example. Suppose the input text is “The cat” and the probability distribution is as shown in the figure to the right. What will the next word be?

###### **Low Temperature**

If we decreased the temperature, we’d narrow the probability distribution, thus increasing the likelihood of higher-probability outcomes and reduce the possibility of lower-probability outcomes. We can see how the already most probable word, “chases”, has increased probability. Lower temperature implies that we get more *deterministic* outputs. Deterministic means that the model tends to produce the same outcomes every time.

###### **High Temperature**

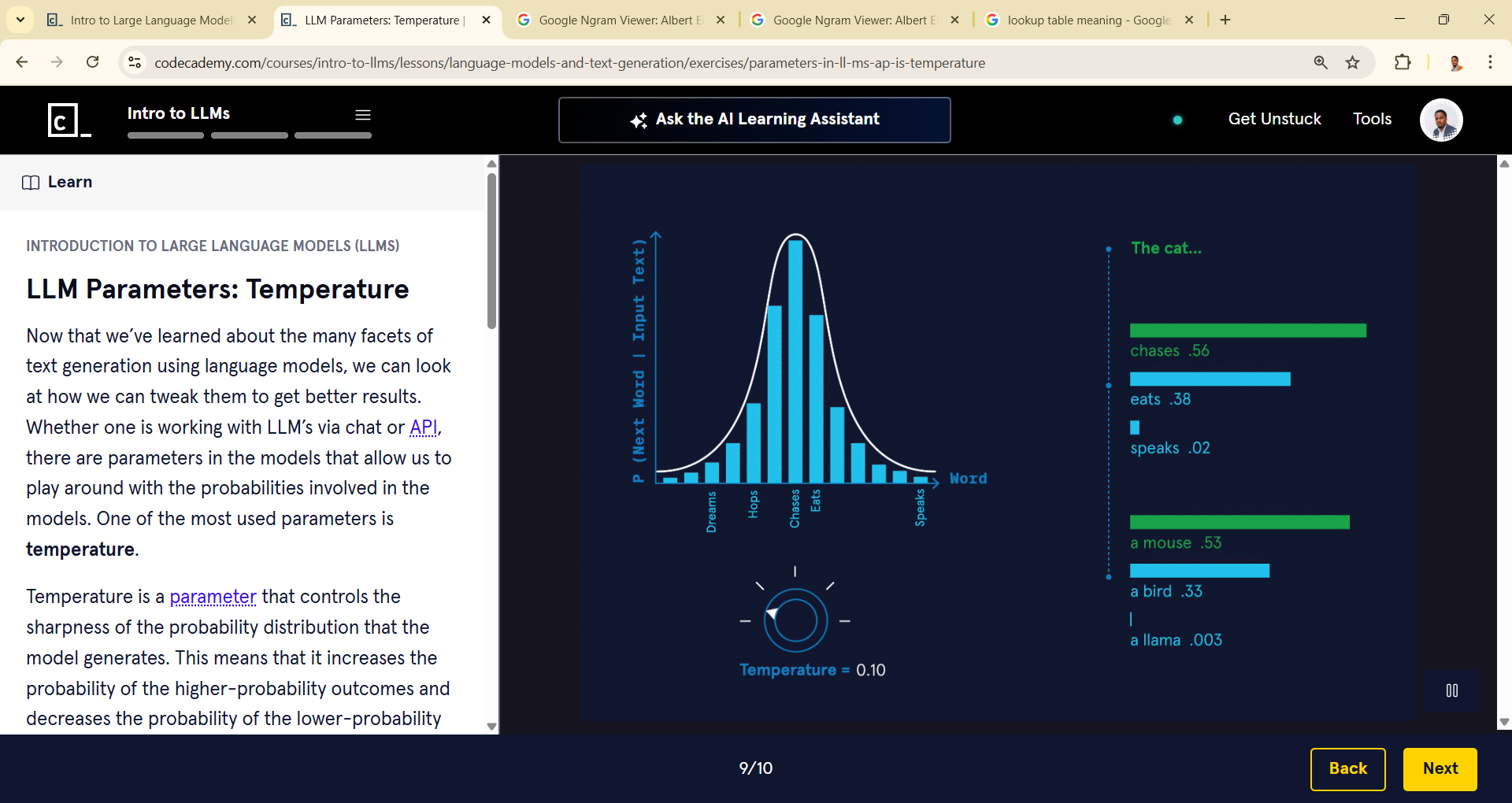
If we increase the temperature, we widen the probability distribution, allowing lower-probability outcomes to become more possible. For instance, “eats” might take precedence over chases. And increasing the temperature further, the previously less probable “eats mango” might become more probable. Increasing temperature might lead to more novel and amusing outcomes. In

[LLM](https://www.codecademy.com/resources/docs/general/llm)

Preview: Docs Loading link description

documentation, the parameter is often described as the one that tunes for “more creative outcomes” or “increased randomness”.

It is important to note that one must always verify that the output of an LLM is grounded in facts. **A high probability does not indicate that the outcome is factually grounded.** Thus, low temperature is no guarantee that the model’s output is correct! Many other parameters in an LLM allow us to play around with the frequency of the outputs, sampling, etc., and you can learn about them in our [LLM API course](https://www.codecademy.com/enrolled/courses/intro-to-open-ai-gpt-api).



**Introduction to Large Language Models (LLMs)**

### **Summary**

LLMs are the technology at the heart of the most recent advancements in the field of NLP. They’re the reason why chatbots come such a long way since the days of ELIZA! Some examples of popular LLMs are GPT, PaLM, Llama, BLOOM etc. These are known as base or foundational models upon which specific applications are built. For instance, the chatbot ChatGPT is built on top of GPT (version 3.5). And the chatbot developed by Google known as Bard is built on PaLM (Pathways Language Model).

So what goes on inside a

[LLM](https://www.codecademy.com/resources/docs/general/llm)

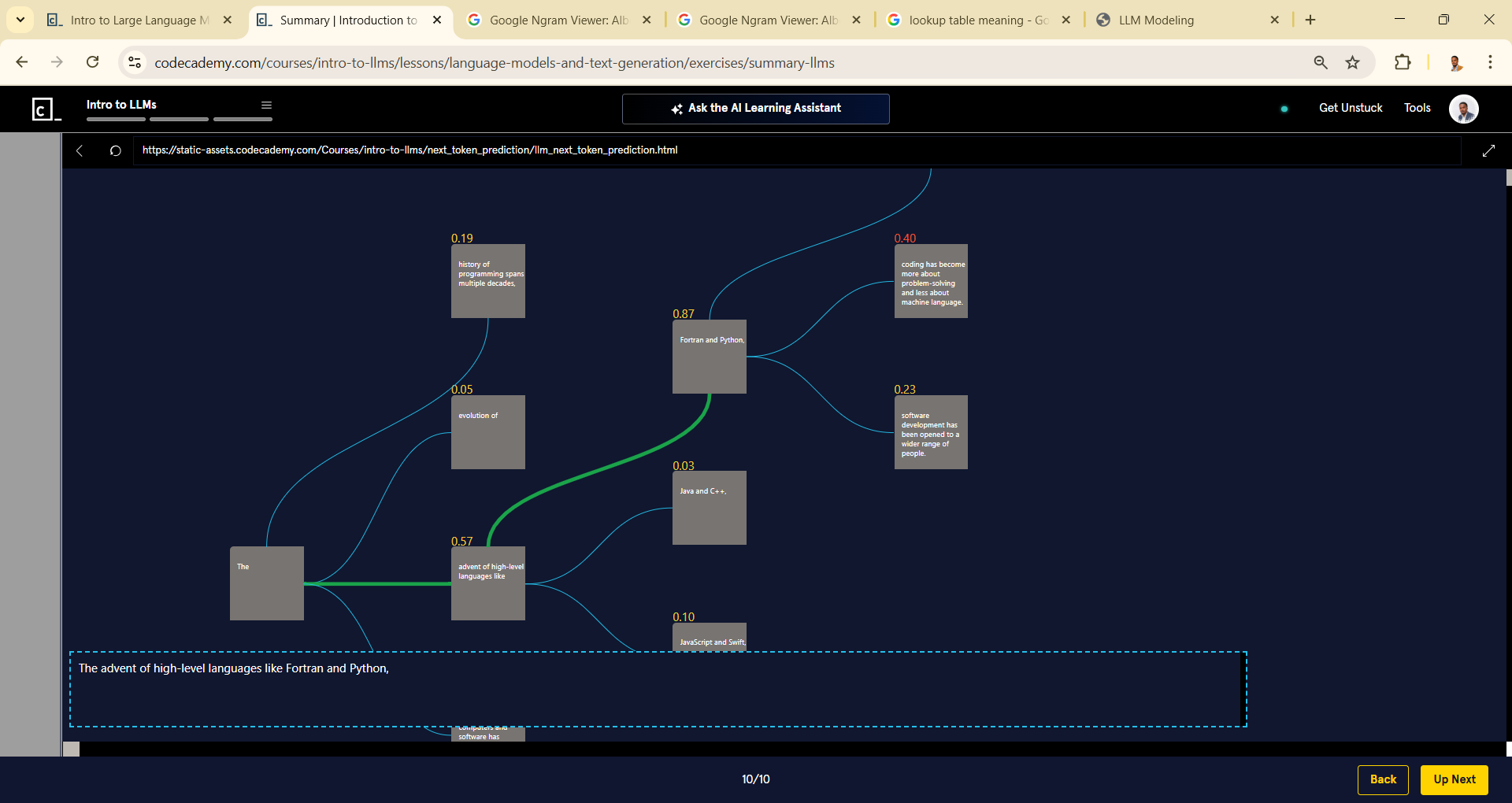
Preview: Docs Loading link description

when it’s prompted? Something like what’s depicted in the applet to the right! The applet shows the multiple pathways of text generation possible at each instance for the LLM. At every step, the best thing to say is chosen based on a set of probabilities generated from its training phase. Feel free to zoom in and/or scroll around the applet to view text generation in action!

To conclude the lesson, here’s recap of all the key concepts you’ve learned about LLMs:

* **Language models** are algorithmic systems trained on a corpus of text to perform a variety of tasks relating to language such as text generation, summarization, translation, etc. Large Language Models (LLMs) are trained on large amounts of text and use neural networks to learn the underlying distribution of words in text.
* LLMs have grown in performance and popularity over the last decade due to a specific type of highly efficient neural network architecture known as the **transformer**. Neural networks solve the problems of lack of generalizability and the curse of dimensionality by compressing the text data into a mathematical representation.
* **Compression** and **generalization** are the heart of the mechanism that makes LLMs powerful tools that can exhibit emergent properties. The downside to generative language models there is no guarantee of factual grounding in their outputs.
* All autoregressive models, count-based or neural, produce outputs based on probability distributions they generate for the next best thing to say. LLM API’s have a variety of parameters to tweak these probabilities to get a desired output. We explored one such important   
  [parameter](https://www.codecademy.com/resources/docs/general/parameter)
* Preview: Docs Loading link description
* , *temperature* that controls for how deterministic the LLM output is.

<https://static-assets.codecademy.com/Courses/intro-to-llms/next_token_prediction/llm_next_token_prediction.html>



The type of neural network that’s used in the GPT language models is: Answer Choices RNNs (Recurrent Neural Networks) LSTM (Long Short Term Memory networks) Transformer CNNs (Convolutional Neural Networks)

###### **ChatGPT said:**

The correct answer is:

✅ **Transformer**

GPT models are built using the **Transformer architecture**, specifically the **decoder-only** part of the Transforme