CHAPTER

1

Introduction to Generative AI

In a matter of just a couple of years, generative AI (GenAI) has gone from a promising new area of AI research to an essential tool that's powering an ever-increasing number of applications across every type of creative and knowledge work that people do. The research firm Gartner predicts that GenAI could consume as much as 3.5% of the world's electricity by 2030. That such a massive investment is being made in this still-young technology is a testament to its potential and usefulness.

When you think of GenAI, you likely think of chatbots like ChatGPT or image generation and manipulation applications like Midjourney and Adobe Firefly. However, it's important to recognize the difference between the AI model (such as OpenAI's GPT-4o) and an application that uses the model (such as ChatGPT).

Applications are a necessary layer between the user and a GenAI model, and they can make all the difference in whether the outputs of the model are mostly nonsensical predictions of the next word in a sentence or a convincing conversation.

Creating applications that make use of AI, along with the processes and systems that support them, is sometimes called *AI engineering*. GenAI models and the application programming interfaces (APIs) that the creators of the models have made available are enabling AI engineers to create software that they could only have dreamed of before GenAI.

In this chapter, you learn about the history of AI-powered applications, the basics of how large language models (LLMs) work, and why GenAI has become so important for developing applications.

Evolution of AI Applications

The history of the vast field of computing that we now call artificial intelligence (AI) goes back to at least 1950, when an English mathematician and computer scientist named Alan Turing wrote "Computing Machinery and Intelligence." In his paper, Turing introduced what became known as the *Turing test*. The idea of the Turing test was to attempt to define a standard for a machine to be called "intelligent." Turing suggested that a computer could be said to "think" if a human conversing with it couldn't tell it apart from a human being.

The first attempts at mimicking human intelligence and behavior using a machine were created before the invention of modern electronic computing. They were bound by the limited processing power of mechanical computers or vacuum-tube-based computers. After transistor-based computers and microchip-based computers were introduced, the potential for developing useful AI applications began to become a reality.

In just seven decades, AI went from being a theory in the mind of a genius to something that some people are genuinely worried will soon replace many forms of human work. To help imagine and map what AI applications will look like and do in the future, let's take a brief look back at the history of AI.

Key Eras of AI Development

Several strategies for creating machines that can think as humans do have been tried over the years. These strategies include rule-based systems, statistical learning, and neural networks. At each step, applications of the technology were created that proved its value, and at each step, researchers were disappointed to find that making the type of progress that would make AI capable of passing the Turing test was far more difficult than they had expected.

The following sections outline the rough progression of AI from its early days to the current state of the art, without getting into the technical weeds (yet) of describing different algorithms and types of machine learning (ML) models.

Logic and Rules-Based Systems

The first attempts at creating AI applications used predefined rules and logical reasoning to make decisions. For example, in 1951, Christopher Strachey, an English physicist and mathematician, created a draughts game (checkers, to Americans) for the Ferranti Mark 1 computer at the University of Manchester.

Due to hardware issues and coding errors, initially it didn't work correctly. However, by 1952, Strachey had improved the program, and it could play a complete game.

Early Machine Learning

At the same time as rules-based AI was taking shape, the first experiments in ML were also going on. In 1952, Marvin Minsky created the first artificial neural network (ANN), named the Stochastic Neural Analog Reinforcement Calculator (SNARC), which used a computer with 300 vacuum tubes to simulate a network of 40 neurons. Also in 1952, Arthur Samuel developed Samuel Checkers, which was the first AI checkers player designed to be self-learning.

Optimism for the potential of AI was running high, and at a workshop held at Dartmouth College in 1956, attendees predicted that we'd have machines as intelligent as humans within a generation. However, by the early 1970s, it became clear that achieving intelligent machines wouldn't be so easy after all. In 1974, the British and U.S. governments cut off their funding for AI research. The period between 1974 and 1980 is known as the "first AI winter."

Expert Systems

In the 1980s and 1990s, AI application development was reinvigorated by rule-based programs called *expert systems*, which were being successfully deployed in a variety of industries, including manufacturing, medical diagnosis, customer service, transportation, law, and financial services. Designed to augment human decision-making, expert systems consist of four parts:

- A knowledge base created from facts (gathered from human experts) about the expert system's subject area
- A working memory that stores data about the current problem
- An inference engine, which uses information from the knowledge base and a set of rules to solve new problems
- A user interface that accepts data from and outputs data to a user, who may or may not be an expert

But, once again, the initial enthusiasm for AI waned, and by the end of the 1980s, the second AI winter had begun.

Big Data and Statistical Machine Learning

The second AI winter trudged along. But in the meantime, the rise of the internet in the late 1990s and 2000s was leading to an explosion of easily available data. Sources of large datasets include millions of public websites; data published

by governments, researchers, and nonprofit organizations; financial data; data generated by social media websites; and sensor data from internet of things (IoT) applications.

During the "era of big data," as it's called, the exponential increase in the variety of data formats and the velocity of new data being generated created problems and opportunities for data analysts. Fortunately, computer hardware and methods for distributing data and compute power across multiple computers advanced substantially, which prompted researchers to take another look at ML.

AI researchers turned their attention to statistical approaches to machine learning. Statistical machine learning (SML) utilizes mathematical methods to create models designed to draw conclusions and make inferences from large datasets.

Uses for SML include various data analysis tasks such as categorization, clustering, prediction, and regression. SML is also used for computer vision and speech recognition.

Deep Learning

By the early 2000s, SML had demonstrated its applicability across various domains. However, this approach necessitated considerable and costly efforts to define the features of data that the model would learn—a process known as *feature engineering*. Researchers recognized that automating this process could broaden the utility of ML across a wider range of applications.

The increase in computational power enabled the training of larger and more complex neural networks. Additionally, datasets such as ImageNet provided the extensive labeled data required for effective network training. This combination of high computational power, rich datasets, and the objective of reducing manual intervention facilitated the emergence of deep learning as a prominent approach in AI.

Deep learning, a subset of ML, employs layers of artificial neural networks to identify patterns in data. A key attribute of deep learning is its ability to enable models to automatically learn hierarchical features from raw data, rather than relying on manually engineered features from structured data. Prior to deep learning, defining relevant features such as edges, color histograms, or keywords was typically necessary to train a model to recognize objects or sentiment. In contrast, deep learning models autonomously learn these features during training by optimizing the parameters of each layer to minimize error on the task at hand.

Before 2011, the performance of neural networks was not superior to other ML methods, leading most researchers, particularly in computer vision, to doubt that deep learning would surpass manual feature engineering. However, deep learning underwent a revolution in the 2010s, largely due to the utilization of specialized hardware involving graphics processing units (GPUs). In 2012, Alex

Krizhevsky, in collaboration with Ilya Sutskever and his Ph.D. advisor Geoffrey Hinton at the University of Toronto, developed AlexNet. This neural network applied deep learning to classify images of objects and won the ImageNet Large Scale Visual Recognition Challenge by performing 10.8% better than any competitors, sparking renewed interest in deep learning.

Advancements in deep learning have led to significant improvements across various tasks, including speech recognition, natural language processing (NLP), computer vision, and image generation. Table 1-1 presents some of the major AI milestones leading up to deep learning and milestones that resulted from the use of deep learning.

Table 1-1: Major AI milestones (1950–2022)

YEAR	EVENT	DESCRIPTION
1950	Alan Turing publishes "Computing Machinery and Intelligence"	Proposes the Turing test, initiating the philosophical basis for AI.
1951	Christopher Strachey's draughts program	One of the first AI programs; ran on the Ferranti Mark I and played checkers.
1952	Marvin Minsky designs SNARC	First neural network simulator; modeled a rat learning a maze.
1956	Dartmouth Workshop	Term "artificial intelligence" coined; field formally founded.
1957	Perceptron algorithm (Frank Rosenblatt)	Early neural net for binary classification.
1959	Arthur Samuel's checkers program	Early reinforcement learning; improved by self-play.
1966	ELIZA (Joseph Weizenbaum)	Early chatbot simulating a psychotherapist.
1970	SHRDLU (Terry Winograd)	Demonstrated natural language understanding in a block world.
1980	Expert Systems (e.g., XCON)	Commercial success for rule-based reasoning systems.
1987	Second AI winter begins	Drop in funding due to unmet expectations.
1997	Deep Blue defeats Garry Kasparov	First AI to beat world chess champion.
1998	LeNet (Yann LeCun)	Neural network for digit recognition; precursor to deep learning.
2005	Emergence of Big Data	Web-scale data and computing enabled ML advances.
2012	AlexNet wins ImageNet	Sparked deep learning boom with GPU-accelerated neural networks.

Continues

Table 1-1 (*continued*)

YEAR	EVENT	DESCRIPTION
2014	GANs introduced (Ian Goodfellow)	Enabled realistic generative modeling using adversarial training.
2016	AlphaGo defeats Lee Sedol	Breakthrough in reinforcement learning and strategy games.
2017	Transformer architecture introduced	Revolutionized NLP, which is the ability of computers to understand and communicate using human language.
2020	GPT-3 released (OpenAI)	175B parameters; enabled few-shot and zero-shot learning.
2020	DeepMind AlphaFold 2	Solved a 50-year grand challenge in biology by predicting protein folding with near-experimental accuracy.
2021	DALL·E and CLIP released (OpenAI)	Advanced text-to-image and vision-language alignment.
2022	ChatGPT released (GPT-3.5)	Made conversational AI widely accessible and user-friendly.

The Rise of Generative AI

The early 2020s saw an AI boom, which was made possible by the combination of big data, deep learning, and innovations in LLMs. GenAI enabled the creation of applications such as AI chatbots, text-to-image generators, audio and voice generators, and video generators.

Unlike previous ML models, which were generally designed for a specific purpose, the models that emerged in the early 2020s were trained to be useful for a wide range of applications. Models like OpenAI's GPT models, Google's BERT, and Anthropic's Claude pioneered the class of AI models called *foundation models*. Foundation models can respond to prompts and generate content about seemingly any topic.

WHAT IS A GPT? **GPT stands for *generative pretrained transformer*. Although the acronym GPT is most commonly associated with OpenAI models, it also describes the architecture used by other LLMs, such as Google's Gemini, Meta's LLaMA, and Anthropic's Claude. These models follow a similar strategy: they are first pretrained on massive amounts of text and later fine-tuned for specific applications. *Transformer* refers to the neural network architecture, which you learn more about in the section of this chapter titled "Large Language Models."**

The EU defines a foundation model as an "AI model that is trained on broad data at scale, is designed for generality of output, and can be adapted to a wide

range of distinctive tasks." Because they've been trained to be broadly useful and can be trained or instructed to be useful in cases outside of subjects in their training data, the range of applications for foundation models is unlimited.

For example, if you're developing a new product or a new piece of software and you want to make an AI chatbot available to your customers, you can supply a foundation model with your product's documentation, and it will be able to answer questions related to the new data even though it wasn't in the original training data for the model. Compared to traditional AI models, which are designed to be useful in a limited set of cases—weather, economics, or translation, for example—foundation models learn from and take into consideration a broad set of experiences. The well-roundedness of foundation models makes them seem much more human than previous types of AI models.

Because GenAI models simulate the way humans interact much more closely, however, they are subject to the same sorts of flaws and quirks as human-to-human communication. These limitations include a tendency to fabricate data (aka *hallucinate*) and the potential to be biased toward things they have more experience with.

NOTE **The popular term for false or misleading data presented as factual by a GenAI system is *hallucination*. I prefer the term *fabrication*, to avoid attributing human characteristics to (or anthropomorphizing) ML models. However, I will occasionally use the term *hallucination* because it's the more commonly used word.**

Like every AI model that came before, GenAI models are just algorithms. An algorithm is a set of rules or a process (like a recipe) to be followed to solve a particular problem. As algorithmic computing systems, GenAI models lack firsthand knowledge of being human and emotions. Depending on the application, the lack of human experience and emotions can be positive or negative, but it must be considered when developing applications based on AI models.

Transition to GenAI

The interesting thing about the history of AI from the early experiments in the 1950s to the latest generative models is the evolution from task-specific models to general-purpose generators. For example, the first talking machines were amusement automatons such as mechanical fortune tellers that relied on randomness and people's gullibility. This led to the ELIZA chatbot. Originally intended as a joke, ELIZA used pattern matching and text substitution to give somewhat convincing canned responses that mimicked a conversation with a therapist. ELIZA led to rules-based customer service chatbots that used a database of rules and information and could be customized for any purpose by supplying it with different rules. This led to general-purpose bots like Siri and Alexa that are still mostly rules based but can also interact with devices

and perform many different tasks within their domain. The latest advance is chatbots that can consider context and simulate natural language so well that they're outperforming humans on standardized tests like the SAT and in coding challenges.

Just as the invention of space travel didn't make automobiles obsolete, GenAI won't completely replace rule-based, logic-based, or statistical AI. There are many examples in which task-specific models will continue to outperform general-purpose models. For example, using a generative model for spam email detection is a waste of resources, and it's likely that any benefit in accuracy gained from using the latest GPT model will be far outweighed by the increased cost and loss of performance. The same goes for checkers and chess applications. It may be that a new generative model with reasoning capabilities can do well in playing chess or identifying spam, but it won't be able to play chess at the level of the latest chess engines, and it won't be as fast and inexpensive as a dedicated spam detection system.

Generative models have been integrated into all sorts of consumer and enterprise applications. In many cases, the integration makes sense and makes the software more useful or user-friendly. In some cases, however, the integration of GenAI into products was done for no other reason than because it was easy to do, and it enables a business to advertise that its product uses artificial intelligence.

One example of AI integration that I've seen and been particularly irked by was the setup wizard for a new printer. Connecting a printer to a network and to your computer isn't a creative task. However, the maker of this printer instructed a chatbot to walk users step by step through configuring the printer. Although the idea of having an expert on call to help set up a new printer may be a good one, the implementation was horrible for the following reasons:

- Using the setup chatbot wasn't optional, leaving anyone who has ever set up a printer and doesn't need the wizard frustrated.
- The touchscreen for the printer is small and imprecise.
- The chatbot's responses were often too lengthy to fit on the screen, requiring the user to use the tiny and imprecise scrollbar on the left of the tiny screen to read the whole message.
- The chatbot didn't provide any way for the user to ask questions.
- The instructions given by the chatbot were wrong or for a different model printer.

Simply using GenAI in an application doesn't necessarily improve the product. As in the case of the printer setup wizard, it may have the opposite of the intended result, making it more frustrating to use or even rendering a product useless in some cases.

Integrating GenAI into software or websites has become incredibly easy. Many of the creators of GenAI models make simple web APIs or SDKs publicly available to developers. As you'll see in Chapter 3, with a few lines of code, you too can advertise that your application features AI. If your application is a tool for the creation of some form of content, it's possible that at least some of your users will appreciate the new features. However, if your application is a utility (such as a video format conversion tool, for example) or a tool that your users expect to work correctly every time, integration of AI may be a terrible idea.

Some of the factors to consider when deciding whether to integrate GenAI into an application include the following:

Real user value. Avoid adding AI to an application just to be trendy. Interview users (or potential users), and focus on solving real pain points or enabling features that the users want.

Fit with the core workflow. GenAI works best when it fits into an application's normal flow rather than being an add-on or gadget in a sidebar.

Privacy and intellectual property (IP). Will integrating GenAI into your application require using private user data as inputs to the model?

Hosted or on-premises? Will your application access a hosted model via an API, or will you host it yourself?

Cost. Hosted models charge for usage based on the combined length of the input and output. Although the rates may seem trivial (amounting to the equivalent of pennies for hundreds of thousands of words), they can get out of control at scale.

Business model impact. How will integrating AI into your application change its perceived value and billing?

Accuracy. GenAI hallucinations can erode user trust. Can you implement filters and other mechanisms to reduce or eliminate the potential for factually incorrect, nonsensical, or inconsistent outputs?

You'll learn more about these considerations in the upcoming chapters.

Understanding AI and ML

Artificial intelligence, broadly defined, refers to the ability of a computer to perform tasks that are commonly associated with the intellectual processes characteristic of humans. *Machine learning* is a subset of AI that deals with enabling computers to learn from data and improve their performance autonomously.

Deep learning is the subset of ML designed to emulate how we think humans learn. In deep learning, data is processed through layers of artificial neurons in an ANN to learn increasingly detailed information about the data.

As AI and ML continue to evolve, their integration into workflows and applications demands thoughtful consideration. These technologies are no longer confined to isolated functionalities; rather, they are reshaping industries by embedding themselves deeply into everyday processes. From predictive analytics that guide decision-making to autonomous systems that perform tasks with precision, AI and ML are heralding a shift toward intelligent automation.

In this section, you learn some important concepts in AI, ML, and LLMs.

What Machine Learning Can Do

ML can be used for a wide variety of tasks, and the task you want the model to perform will determine how you train it. The types of training can be grouped into the following categories:

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning
- Self-supervised learning

Supervised Learning

In supervised learning, you use a labeled dataset. Labeled data uses input–output pairs to train the model. For example, you could train a model to differentiate apples from things that aren't apples. This is called *binary classification*.

To create a binary classification model to pick out apples, you'd first gather as many pictures of apples as you could find. You'd then pair these pictures of apples with the correct answer you want the model to give when it analyzes a picture of an apple. This would likely be the word "apple."

However, if you only train a model on things that are apples, it won't learn anything except maybe that every picture is a picture of an apple. You also need to train it to identify pictures that are "not apples." With a large dataset of things that are labeled "apple" and things that are labeled "not apple," you can then train the model. But you don't use all the pictures from the dataset for training, because you need to hold some back (maybe 30%) to use for testing your model on data it's never seen before, to see how accurate it is.

The other use case for supervised learning is in regression. Regression predicts a continuous value. For example, you can create a model to predict the price a house will sell for based on information about other houses that have sold. By training an ML model on information such as the date, location, square footage, and other facts about known home sales, the model will be able to make predictions about the sales price of homes that haven't sold or when the selling price isn't known.

Unsupervised Learning

Unsupervised learning finds patterns in data without predefined labels. Examples can include any data in any format, such as text, music, video, and images. In unsupervised learning, the model uses algorithms to find similarities in the data that it can use to group pieces of data together.

Using an ML model to discover groups in unlabeled data is called *clustering*. Example uses for unsupervised training include customer segmentation, fraud detection in financial transactions, fault detection in machinery, medical imaging, and artistic style classification.

Unsupervised learning is also used to remove noise in audio and video, to simplify data for use by other ML models, and to create labels for supervised learning.

Semi-Supervised Learning

Semi-supervised learning uses a small amount of labeled data and a large amount of unlabeled data. This type of learning is used when labeling data is expensive and there's a lot of available data. The small amount of labeled data teaches the model what patterns to look for. The model can then extract patterns from the much larger amount of unlabeled data.

The benefits of semi-supervised learning are that it reduces the need for expensive manual labeling of data, it improves performance of the resulting model—especially when working with real-world or noisy data—and it can scale to enormous datasets, such as raw data from the internet. A few of the uses for semi-supervised learning include image detection, medical diagnostics, text classification, and speech recognition.

Reinforcement Learning

Reinforcement learning is when a model is trained by using rewards or penalties over time. Reinforcement learning is used in training AI models to play video games, in teaching robots to perform tasks, in dynamic pricing strategies, and for teaching GenAI models to make better predictions. You contribute to the reinforcement learning of AI models that decide what to show you on social media when you "like" or otherwise engage with posts or ads.

Self-Supervised Learning

In self-supervised learning, the system learns from unlabeled data by creating its own labels from the data itself. Self-supervised learning is especially useful for NLP, computer vision, and speech. By creating its own labels, the system can make predictions about missing data, next sentences, and missing audio or video. Self-supervised training is how GenAI models are trained.

Large Language Models

A LLM is a type of ML model that's trained (using self-supervised learning) on a huge amount of text data to understanding and generate human-like language. LLMs are the models behind chatbots like ChatGPT. They are basically text prediction models: they receive input and use that to predict a way to respond. An LLM's predictions (aka output) are based on the patterns that it learned through training.

It's important for anyone who wants to develop applications that take advantage of LLMs to have a basic understanding of the main components and processes that happen when an LLM receives input (aka a *prompt*).

The main phases of generation are as follows:

Tokenization. Converting the input to numbers

Embedding. Converting the tokens to multidimensional vectors

Transform. Using multiple layers of algorithms to find the meaning of the input

Output. Selecting from the most probable outputs

Tokenization

ML algorithms work with numbers, so the first step in responding to a prompt is to convert it into a numeric representation. When an LLM receives input, it uses a process called *subword tokenization* to split the input into chunks called *tokens* (which are, on average, about four characters long) that are represented by numbers.

For example, consider this quote from Bill Gates: "Generative AI has the potential to change the world in ways that we can't even imagine." This quote contains 19 tokens when tokenized for use by OpenAI's GPT-4o model. You can input any text and see how it gets tokenized for OpenAI's models using OpenAI's Tokenizer tool at `https://platform.openai.com/tokenizer`, shown in Figure 1-1.

Each token is represented in the model by an ID number. For example, a period has a token ID of 13. More common tokens (such as a period or the word "the") have lower token IDs. You can see the token IDs in OpenAI's Tokenizer tool (see Figure 1-2).

Knowing how tokenization works is important when developing applications using LLMs because the APIs you'll use to access the LLMs charge for usage based on the total number of input and output tokens. One of your goals as an application developer is to find ways to minimize the number of tokens your application's AI integration uses.

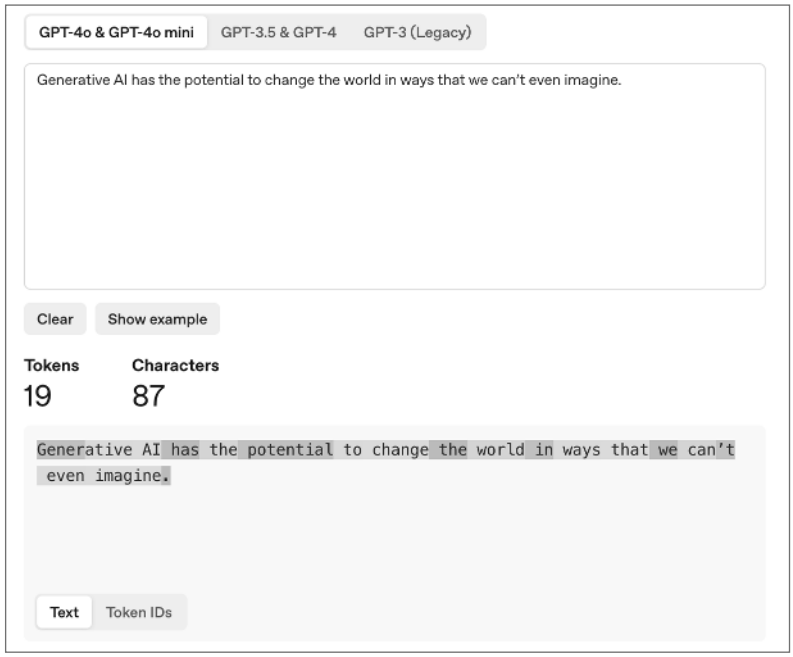

Figure 1-1: OpenAI's Tokenizer tool

Source: Generated with AI using ChatGPT - OpenAI

Figure 1-2: Viewing the token IDs

Source: Generated with AI using ChatGPT - OpenAI

Embedding

Next, the tokens go through a process called *embedding*, in which the tokens are converted into multidimensional vectors (which are like lists of numbers) that represent the semantic meaning of the text.

GPT-3 uses 12,288 dimension vectors. OpenAI hasn't (at this time) disclosed how many dimensions its newer models use. Each of these dimensions stores some piece of information about the token.

You can visualize embeddings more easily by looking at a much smaller embedding. Table 1-2 shows how you might represent different fruits as two-dimensional vectors, where the first vector is "roundness" and the 2nd is "redness."

Table 1-2: Showing fruits as two-dimensional vectors

FRUIT	ROUNDNESS	REDNESS
Apple	0.9	0.8
Banana	0.2	0.1
Strawberry	0.6	0.9
Watermelon	0.8	0.7
Lemon	0.7	0.2
Blueberry	0.95	0.1
Cherry	0.85	0.95
Grape	0.9	0.4
Peach	0.85	0.6
Pineapple	0.4	0.2
Orange	0.9	0.5
Raspberry	0.7	0.85

If you had more dimensions, you could capture more information about each fruit. But even with this very small embedding, you can see that some fruits are more similar than others, as shown in Figure 1-3.

The dimensions in a GPT model's embeddings don't map cleanly to human-interpretable concepts like "roundness" or "redness." Instead, each dimension captures abstract features that the model learned in its training, such as

- Syntactic patterns (e.g., tense, part of speech, sentence structure)
- Semantic relationships (e.g., similarity between "king" and "queen")
- Contextual nuances (e.g., sarcasm, sentiment, topical cues)
- Long-range dependencies (e.g., keeping track of subjects over paragraphs)

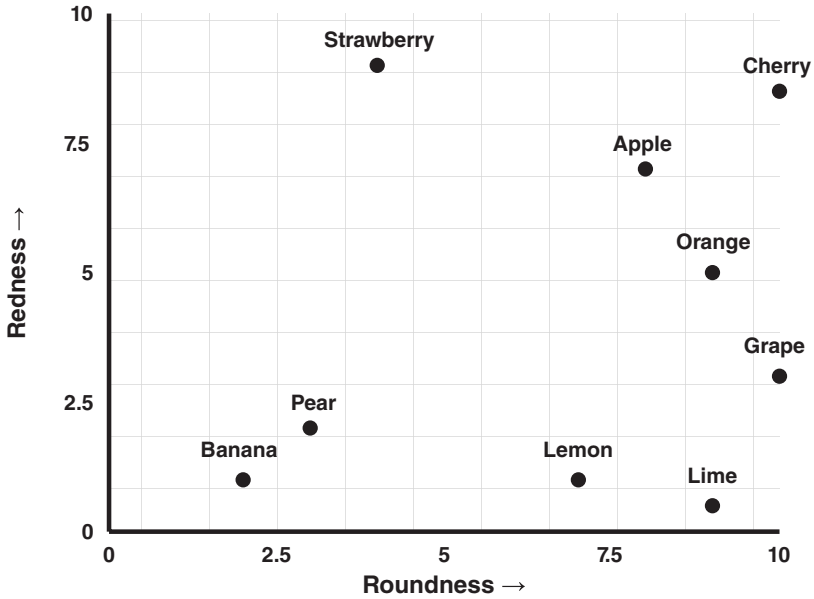

Figure 1-3: Showing fruits as two-dimensional vectors

No single dimension represents just one idea; rather, meaning is spread across the full vector space. You can't point to "dimension 384 = past tense." However, clusters or directions in the space often capture meaningful relationships (like word analogies or category groupings).

Many words, such as "bark," have different meanings depending on the context in which they're used. At the tokenization and embedding phase of generation, however, both kinds of bark (the sound a dog makes and the bark on a tree) have the same values. It's during the next phase that the contextual meaning of a token is figured out.

Transformer Layers

Tokenization and embedding are both done during the input preprocessing phase of generation. Once input has gone through this process, it progresses to the transformer layers. The transformer layers use a mechanism called *attention* that figures out the meaning of the token from the surrounding tokens.

What made all the difference in transforming LLMs from older, much slower models to the models we have today is a mechanism called *self-attention*, which was described in a 2017 paper titled "Attention Is All You Need" (`https://arxiv.org/abs/1706.03762`).

Self-attention is a mechanism where, for each token, the model looks at all other tokens in the input and computes weighted importance scores (attention weights) to decide how much each other token should contribute to its representation.

Prior to self-attention, each token would be evaluated sequentially. This capability makes transformers especially good at handling long-range dependencies in text and is one of the reasons they have become the foundation for nearly all modern LLMs.

Prediction

The result from the transformer layers is a probability distribution, or range, of next token predictions. For example, if the input to the model is "Mary had a little," the model will predict that "lamb" is the most likely next token, because of the extremely common nursery rhyme that begins with this phrase. But it may also predict other, less likely, tokens, such as "cat" or "problem." The token that the model outputs will be one of those from the probability distribution. The model will select the next token from the probability distribution through random sampling. The use of randomness in selecting the final output token is what introduces randomness and creativity into outputs from a generative model.

What Makes Generative AI Different?

Although they're still based on mathematics and algorithms, the outputs from GenAI models are very different from the outputs of previous AI models. As a result, creating applications for GenAI is more like designing jobs for humans to do than traditional software. Some of the ways GenAI is different include these:

- Ability to generate novel content (text, images, code, media)
- Unpredictability and nondeterminism of outputs
- Dependence on probabilistic language
- Requires prompt design, not just data pipelines or logic trees
- Often operates across multimodal domains (e.g., combining text and vision)
- Shifts user experience (UX) expectations to emphasize collaboration, exploration, and ambiguity
- Greater need for guardrails, feedback loops, and user control mechanisms

Generating Content

Prior to GenAI, AI models were mostly used for classifying data and predicting values. For example, a weather model forecasts the likelihood of rain based on current conditions fed into a model trained on historic information. GenAI is also making predictions, but rather than the result being a probability, the result is coherent natural language, images, music, code, or video.

GenAI Is Necessarily Unpredictable

A classification model should be as predictable as possible. If the model is trained to identify possible skin cancer, detect cyber threats, or provide advance notice of dangerous weather conditions, there's no room for creativity.

GenAI models and applications are nondeterministic. This is by design and necessary. A chatbot that always gives the same output when given the same input is of limited use. The trade-off with GenAI is that to become good at communicating, it must be somewhat random, and it will make mistakes.

GenAI Is Probabilistic

GenAI systems, such as LLMs, rely on probability-driven mechanisms to produce output rather than on fixed logic. This probabilistic nature is what allows for creativity and variation, but it also introduces challenges. Table 1-3 shows the difference between probabilistic and deterministic models.

Table 1-3: Deterministic vs. probabilistic models

ASPECT	DETERMINISTIC MODELS	PROBABILISTIC MODELS (GENAI)
Output	Always the same, given the same input	Varies slightly even with identical input
Example systems	Rule-based engines, decision trees, sort algorithms	GPT, DALL·E, stable diffusion
Behavior	Predictable, explainable, repeatable	Creative, flexible, sometimes inconsistent
Strengths	Reliability, auditability, easy to debug	Creativity, adaptability, handling ambiguity
Risks	Rigid, unable to generalize beyond rules	Hallucination, unpredictability, inconsistency
Use case fit	Transaction processing, validation, rule execution	Content generation, dynamic conversations, idea exploration

GenAI Requires Prompt Design

Deterministic models work with deterministic inputs. GenAI, on the other hand, requires creative inputs to achieve the best results. You learn more about designing prompts in Chapter 5.

GenAI Is Multimodal

Not only can GenAI systems respond to prompts about a wide variety of subjects, but they can also respond in multiple modes. The latest GenAI systems

can return images, code, tables, graphs, images, and more in response to text prompts.

GenAI Shifts UX Expectations

GenAI has made the user interface (UI) much less important. Instead of learning a complex system of menus and tools, users can now prompt systems in natural language, as you would a person. UX is as important as it's always been, but the way you think about and design user experience shifts with GenAI systems.

GenAI Needs Guardrails

Because LLMs are probabilistic, it's possible for them to generate responses that are biased, toxic, false, or off-topic for your application. If a user submits prompts to an LLM that contain profanity or hate speech, it increases the chances that the generated content will contain inappropriate language. Such interactions can create legal liability or brand damage. LLMs are also vulnerable to attack by malicious users who attempt to convince the LLM to reveal sensitive corporate data or personally identifiable information (PII).

Model producers add internal safety guardrails to LLMs to prevent these types of incidents from happening as much as possible. However, it's also important for the developers of applications that use LLMs to take measures. These include:

- Fine-tuning the base model to customize the LLM for the specific application
- Using prompt templates to provide structure to the model's inputs and outputs
- Using system prompts to guide the tone of responses
- Adding external guardrails such as forbidden word filters and output validation

You learn more about ethical considerations and pitfalls of GenAI in Chapter 12.

Real-World Examples of AI Integration

AI engineering can take many forms. The following sections present a few examples that you can try to get ideas for how you might use ML, NLP, and GenAI in your own applications.

AI-Enhanced Customer Service Bots

AI customer service bots can dramatically reduce the need for human support agents. Customers often benefit from AI support options because they can ask common questions using natural language and avoid waiting on hold.

Customer service bots must be accurate. For this reason, they may not always be the best fit for using GenAI. This is especially true in the case of industries that handle PII, such as banking, insurance, and healthcare.

Rules-based AI systems that can help with common customer service issues such as shipment tracking, account balance notifications, and appointment setting can be helpful while also minimizing risk.

Figure 1-4 shows Bank of America's customer service chatbot, named Erica. Erica uses NLP, but not GenAI, to understand user requests and respond with a set of predefined answers and actions.

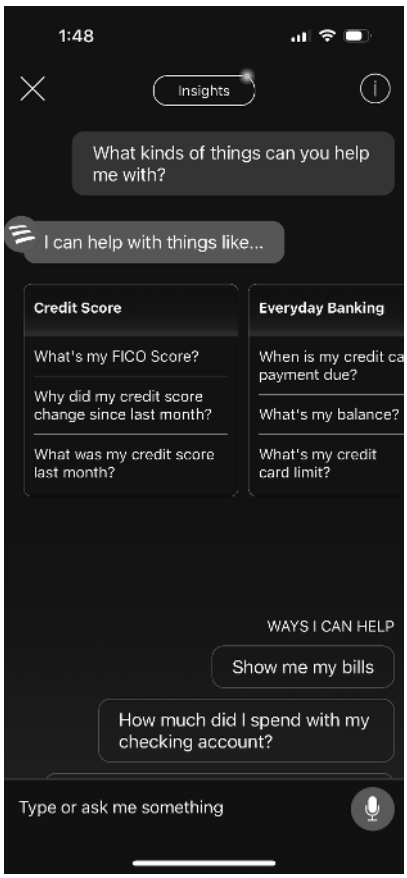

Figure 1-4: Bank of America's Erica chatbot

You learn more about designing and building AI chatbots in Chapter 8.

Generative Writing Tools

Generative writing tools, such as Grammarly and Microsoft 365 Copilot in Word, are designed to help with composing and editing any type of text document.

Because they require a broad knowledge of language and a great deal of creativity, writing assistants are a perfect use of large generative models. However, even when used to generate completely new and original text, such as a story or poem, writing assistants must still be grounded in the rules of the human language's grammar.

Done right, AI writing assistants and AI editing tools can help writers get over writer's block, can act as an intelligent spelling and grammar checker, and can help writers find a better way to say something. However, given too much freedom, AI writing assistants tend to inject tell-tale signs of AI-generated text into otherwise simple emails or paragraphs. Worse still, they will confidently produce fabricated information or citations, which can negatively affect the human using such tools. Figure 1-5 shows how the Copilot 365 writing assistant in Microsoft Word can be used to suggest ways to rewrite text.

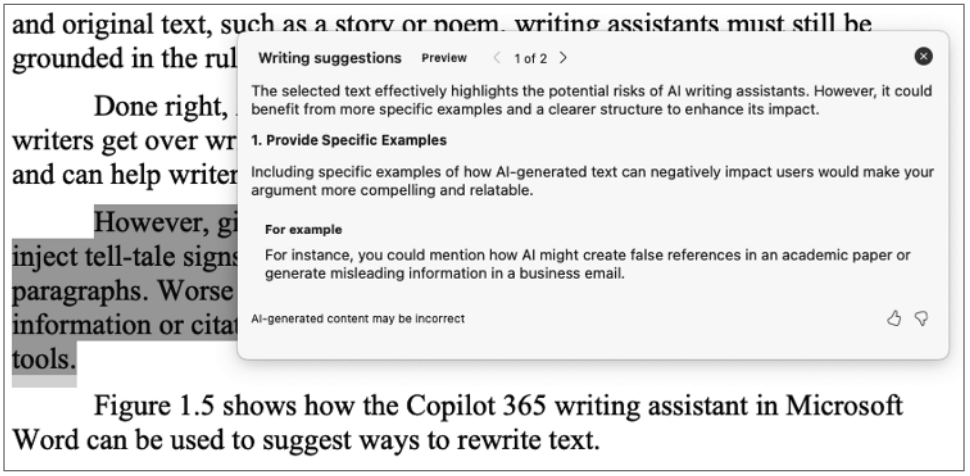

Figure 1-5: Using an AI writing assistant

Image Generation in Creative Tools

AI image generation takes several different forms. Pure text-to-image generators produce unique images by creating an image that aligns with a mathematical understanding of the relationship between images and descriptions of images. If you prompt a text-to-image generator with the phrase "a lovely sunset" and "a beautiful sunset," you'll get completely different results. In fact, if you prompt a text-to-image generator with the exact same prompt twice, you'll get very different results.

Pure text-to-image generation applications, such as Midjourney and image generation in ChatGPT, are unpredictable and therefore of limited use for professional designers and artists. Furthermore, the images that were used to train many

text-to-image generation models come, at least in part, from undisclosed sources, some of which may not have been licensed for training AI models. There is the potential that generating images using a model trained on copyrighted images could be found to be a copyright violation.

Professional image generation tools, such as Adobe Firefly (shown in Figure 1-6), attempt to limit the creativity of GenAI so it can be a useful tool for creating predictable results.

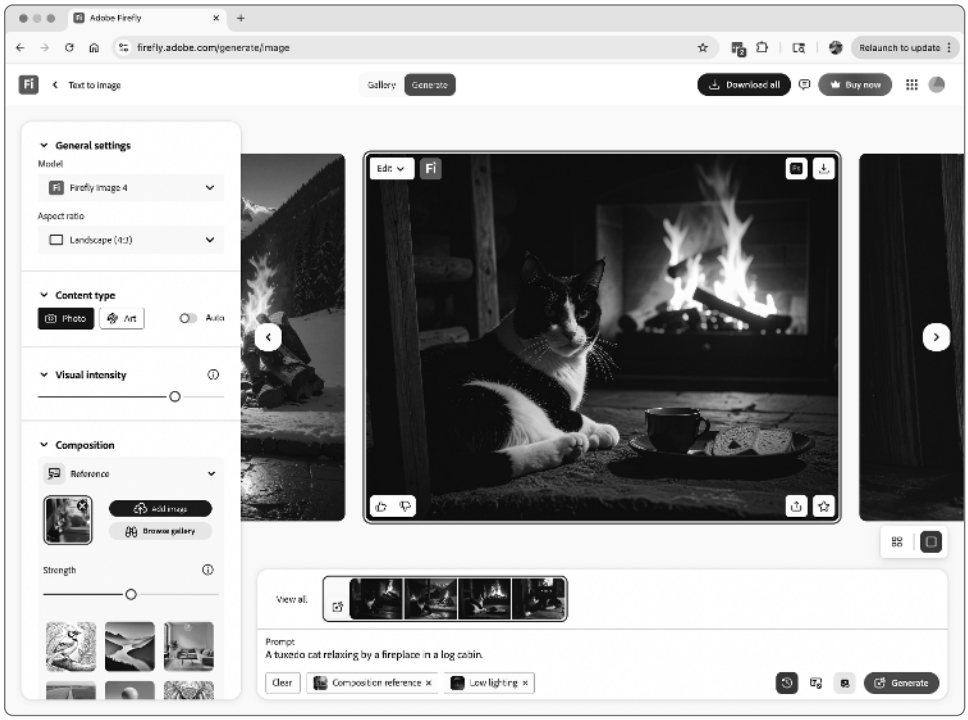

Figure 1-6: Using Adobe Firefly

The image generation process in Firefly starts with configuring attributes of the desired image, such as how photorealistic it should be, the composition, the overall style of the image, the painting or photographic technique to emulate, effects to apply to the image, color, lighting, and camera angle.

Firefly can apply the same settings to multiple images to create a consistent look, and it can even edit existing images to change the style or colors, for example, without changing the composition or subject. Because it's a tool for professional and commercial use, Firefly only trains its models on images that it owns the rights to.

Summary

In this chapter, you learned about the evolution of artificial intelligence from early rule-based systems to modern deep learning and foundation models. We explored the historical milestones that paved the way for generative AI, clarified how machine learning and large language models work, and introduced key distinctions between deterministic and probabilistic AI systems. You also learned why generative AI has such a transformative impact on application design—shifting expectations around user experience, requiring new approaches like prompt engineering, and introducing new challenges around unpredictability and safety.

In the next chapter, you'll dive deeper into the technical features of GenAI models, how they differ from one another, and how to choose the right one for your application.