**1. Introduction to Generative AI**

Aarushi Kansal[**1**](https://learning.oreilly.com/library/view/building-generative-ai-powered/9798868802058/html/610509_1_En_1_Chapter.xhtml#Aff2)

(1)

Melbourne, Australia

Generative AI (artificial intelligence) is a loaded phrase these days. Investors are throwing their money at it, execs are throwing it at each other, and at some point, a manager is probably going to ask you “can we do generative AI too?” or you’re going to get tempted and hack together an LLM-powered bot at 2 a.m. This chapter introduces you, a software engineer, to the booming world of AI, by cutting through all the hype and demystifying AI. I start from the most popular architectures right now and then throughout the book to various models, both open and closed source. I aim to explain these models from the lens of a software engineer as opposed to a data scientist or machine learning scientist. This means the aim is to understand and explain just enough about the foundation models so you can customize and build AI-powered applications, leveraging these models. In particular I’ll focus on diffusion models (you know all those cool AI images you’ve seen on your socials?) and transformer models (think ChatGPT, LLama, music-gen, etc.).

**What Is Generative AI?**

Generative AI is essentially a kind of unsupervised or semi-unsupervised machine learning that allows people to create various types of rich content, like images, text, video, speech, and even music.

With unsupervised learning, a model is able to determine patterns in the data it is fed, often patterns the human eye would simply miss, without needing any kind of labelling. These models leverage neural networks (similar to the networks in our brains) to learn patterns and generate the rich content you’ve been seeing all over the Internet.

Semi-supervised learning is a combination of supervised and unsupervised learning. This means making use of a small number of labelled data (supervised learning) during the training or fine-tuning steps, combined with a large set of unlabeled data (unsupervised learning).

The ability to make use of unsupervised learning on massive amounts of unlabeled data (such as articles, books, images, etc.) is what supercharged companies’ abilities to create massive foundational models such as GPT-4, Stable Diffusion, Llama Bark, etc. Without this style of machine learning, labelling what is essentially all of human knowledge (i.e., the Internet) would have been virtually impossible!

Okay, now that you have a high level intro into generative AI, let’s talk a little bit about different architectures, in particular the two most popular: transformers and diffusion models.

**Model Types**

In this section, we’ll explore two main types of architectures: transformers and diffusion models. While there are a range of architectures, I want to talk to you through the ones the foundation models used in this book are based on. Also, keep in mind, this section is not a deep dive, more of a summary, just enough so you know *what* you’re using, when you build applications on top of these models. This means you won’t be learning the math (but I do recommend you read the papers, research, and understand the math; it’s fascinating!)

First up is transformers and then diffusion.

**Transformers Explained**

Transformers are currently dominating the natural language processing (NLP) space. Most of your favorite models are transformers, for example, GPT-4, Llama, Falcon, etc. Let’s look into transformers and why this architecture becomes so popular. To do that, we need to go through a tiny history lesson.

Once upon a time, there were two main architectures: recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) networks (a type of RNN), specifically designed to handle sequential data (e.g., text). Let’s discuss RNNs and then LSTMs.

**RNNs**

RNNs maintain a memory of previous inputs in their internal structure to process sequences of inputs.

Imagine reading a book and trying to predict the next word in a sentence. If you’re reading word by word without remembering the previous context, it’s tough. But if you recall the earlier part of the sentence, it becomes easier. RNNs do something similar: they remember the “history” to make sense of the current input.

Let’s take a quick look at the basic workflow of an RNN in Figure [1-1](https://learning.oreilly.com/library/view/building-generative-ai-powered/9798868802058/html/610509_1_En_1_Chapter.xhtml#Fig1).

1. 1.

**Input**: At each step, the RNN takes in an input (e.g., a word in a sentence). **(Xn)**

1. 2.

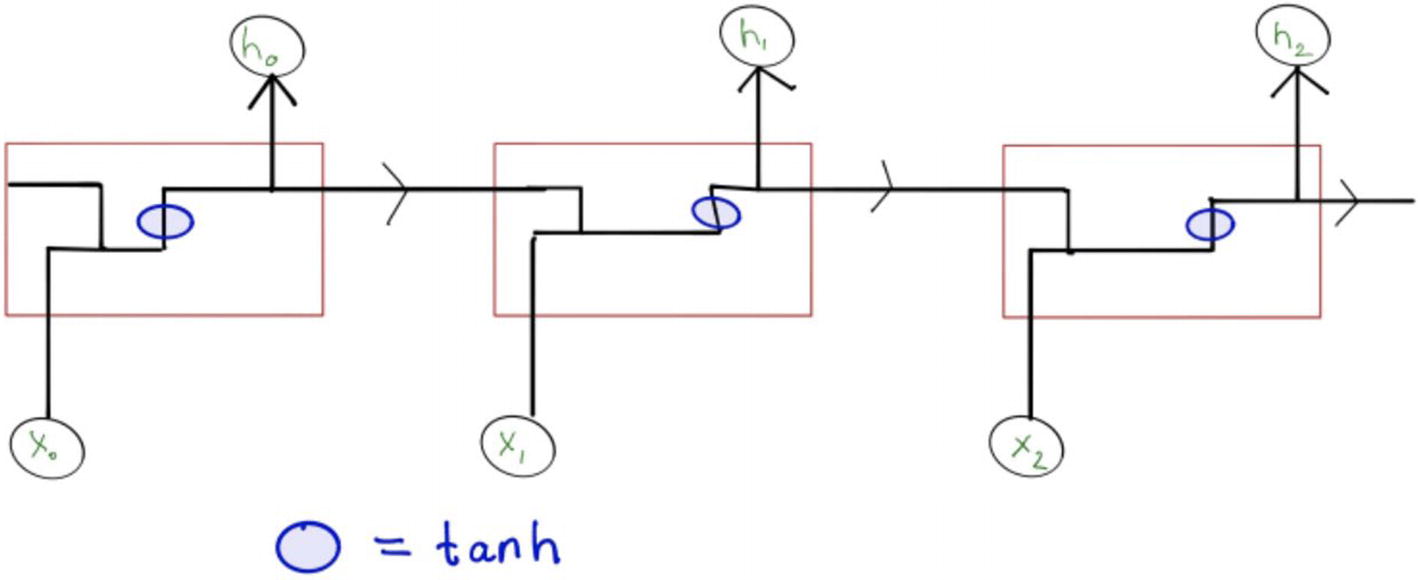
**Hidden State Update**: This input, combined with the previous hidden state **(hn)** (memory), is used to update the hidden state. This new state might carry forward crucial information and forget irrelevant details.

1. 3.

**Output**: Based on the updated hidden state, the RNN might produce an output (e.g., predicting the next word in a sequence). **The output is a combination of Xn and hn**.

1. 4.

**Move to Next Step**: The process repeats for the next element in the sequence.



***Figure 1-1***

RNN architecture

While these basic RNNs are excellent for modelling sequential data like text or time series data, they have the fatal flaw of struggling to remember distant past information. In other words, they have a short-term memory. This tendency to forget is called the vanishing gradient problem.

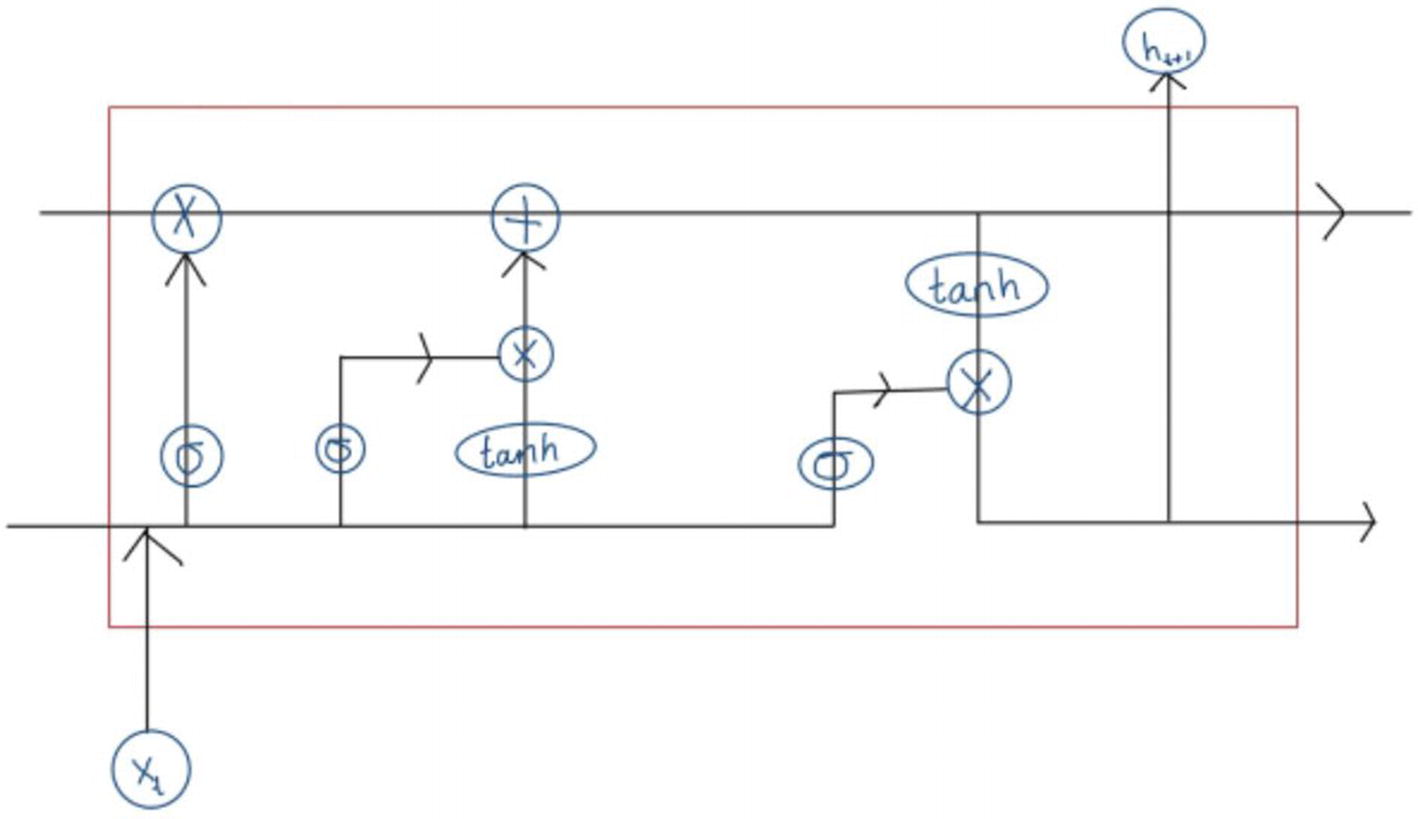
This brings us to LSTMs, designed specifically for long-term memory.

**LSTMs**

The short-term memory problem is addressed by using LSTMs, which have a more complicated structure but function more similarly to how a human might read a book or hold a conversation.

In the previous RNN, you can see that the network is able to remember previous information because we pass the previous hidden state (h) into the current cell. Continuing on from this observation, you can also see how hidden states from further back cells become diluted; essentially the information in those states vanishes.

One of the core observations in Figure [1-2](https://learning.oreilly.com/library/view/building-generative-ai-powered/9798868802058/html/610509_1_En_1_Chapter.xhtml#Fig2) is the top horizontal line, which transfers the vector straight through the cell and through the entire network. This means that information can flow through the sequence, essentially unchanged, meaning this network has the capability to remember information from further behind in the sequence. Kind of like a sushi train, food keeps passing along, and you can remove, modify, or leave the sushi as is.



***Figure 1-2***

LSTM cell

But you also don’t want to just pass information along with no modifications. The way that humans understand and process information is based on our ability to place more or less emphasis on different parts of a sentence or paragraph, based on context or prior knowledge.

To reproduce this kind of behavior, LSTMs use gates (forget, input, and output specifically) to determine what action to take.

So the basic workflow goes like this:

1. 1.

**Input Vector:** Similar to RNNs, the LSTM unit takes in an input vector and the previous hidden state at each time step.

1. 2.

**Gates in Action:**

* + The **forget gate** decides which parts of the cell state to throw away.
  + The **input gate** decides which values to update in the cell state.
  + After these updates, you have the new cell state that carries long-term memory.
  + The **output gate** determines what the next hidden state (short-term memory) should be.

1. 3.

**Output:** The LSTM produces an output, which is the hidden state passed to the next LSTM unit in the sequence.

1. 4.

**Move to Next Step:** The updated cell state and hidden state are passed to the next LSTM unit in the sequence, and the process repeats.

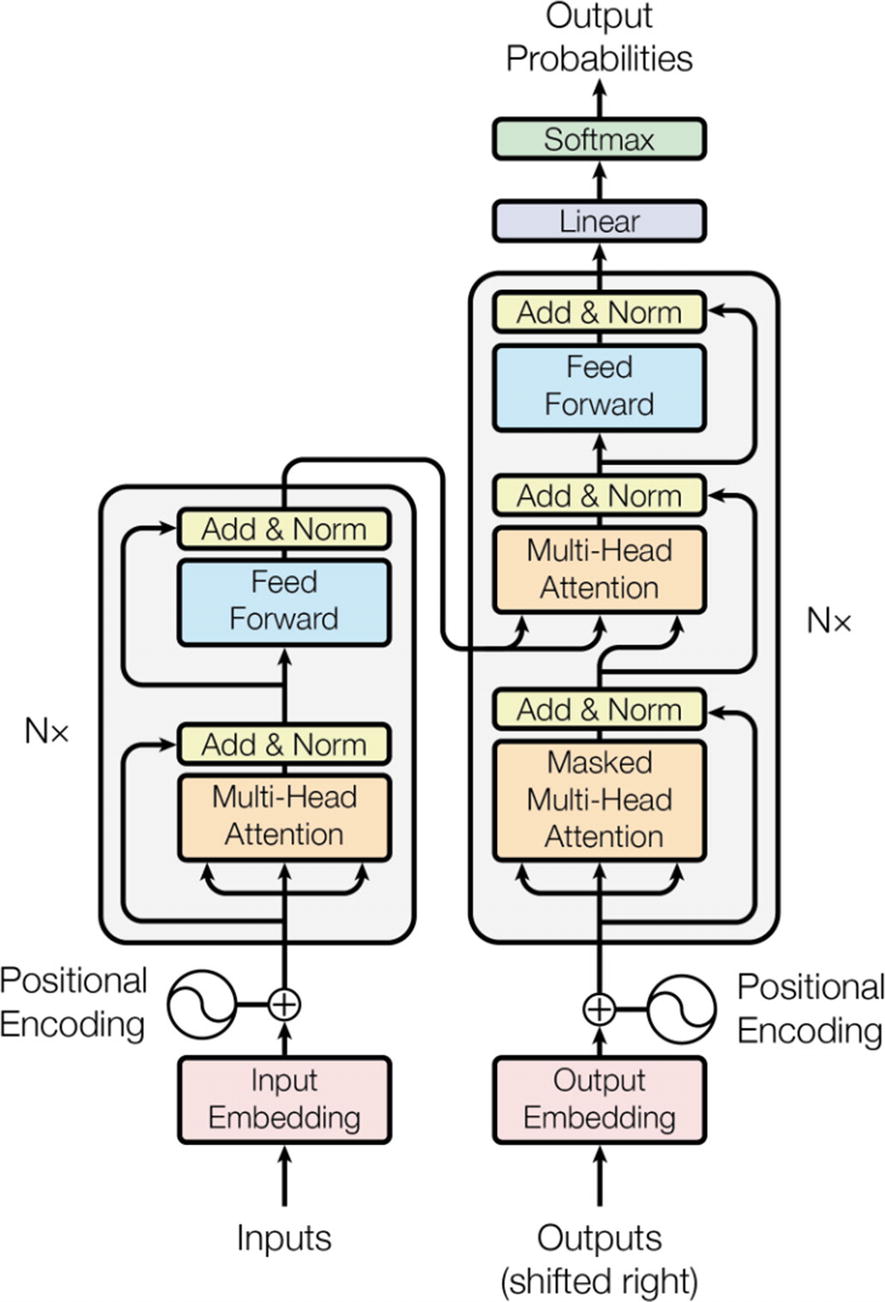
So with this variation of an RNN, you get an *improvement* on the vanishing gradient problem, but it’s still not entirely solved. LSTMs remember for longer but not quite long enough.

**Transformers**

Fast-forward to 2017, a groundbreaking paper named “Attention Is All You Need” was published, with the key creation of a self-attention mechanism.

These models are able to track relationships between words and concepts and understand “context” in language – kind of like we as humans do instinctively, without even actively having to think about it. When humans talk about context, what we mean is attention. For example, when you’re translating a piece of text from English to Spanish, you’ll likely need to pay attention to words, not just next to each other, but distant from each other, because they can change the meaning, the tense, conjugation, and overall form of a word. Attention in the context of transformers is very similar. In other words, ensuring a neural network is able to glean context, because context heavily influences words in almost all NLP tasks.

Let’s take a look at the overall workflow shown in Figure [1-3](https://learning.oreilly.com/library/view/building-generative-ai-powered/9798868802058/html/610509_1_En_1_Chapter.xhtml#Fig3).



***Figure 1-3***

Transformer architecture Image source: [https://​arxiv.​org/​pdf/​1706.​03762.​pdf](https://arxiv.org/pdf/1706.03762.pdf)

1. 1.

**Input Representation**

**Tokenization and Embedding**

* + First, your raw input (some text) is tokenized. This means breaking down the input into chunks – these could be words or characters.
  + Then each token is mapped to a vector using an embedding layer. This vector representation captures the semantic meaning of the token. This is essentially representing words + meanings in numerical form.

**Positional Encoding**

* + Since the transformer doesn’t process data in sequence like RNNs, it doesn’t have an understanding of order of tokens. So a positional encoding is added to the embeddings. This makes sure the model can account for the position of words in a sequence.

1. 2.

**Transformer Layers**

The core of the transformer model consists of a stack of identical layers. Each layer has two main components:

* + Multi-head Self-Attention Mechanism
  + Feed-Forward Neural Network

**Multi-head Self-Attention**

* + This mechanism allows the model to focus on different parts of the input sequence when producing an output for a particular token.
  + The “multi-head” part means this attention process happens in parallel multiple times, allowing the model to focus on different semantic aspects simultaneously.
  + The attention mechanism uses three weight matrices called Query, Key, and Value, which help in determining how much attention to pay to various parts in the input sequence.

**Feed-Forward Neural Network**

* + Each attention output is passed through a Feed-Forward Neural Network (separately but in parallel). The same network is applied to each position.

**Residual Connection and Normalization**

* + After both the attention and feed-forward stages, there’s a residual connection that helps in training deeper networks.
  + Layer normalization is also applied after adding the residual connection.
  + The residual connection helps with the vanishing gradient problem, and layer normalization aids in faster and more stable convergence.

1. 3.

**Output**

* + If you’re using just the encoder part (like BERT), the output is typically a vector representation of the entire sequence or individual tokens, which can be used for tasks like classification.
  + If you’re using the decoder part (like GPT-4), the output is another sequence, which is the result of transforming the input sequence.

1. 4.

**Additional Components**

**Masking**

* + In certain situations, like training a language model, you don’t want certain words to pay attention to future words in the sequence (because they shouldn’t be “known” yet). Masking makes sure the model is “blind” to these future tokens during training.
  + This is crucial for training models like BERT, where you want to predict a masked-out word without “cheating” and looking at it. For GPT, the masking makes sure that when predicting a token, the model can’t look at future tokens.

**Final Linear and Softmax Layer (For Tasks Like Language Modelling)**

* + The decoder’s output can be passed through a final linear layer followed by a softmax to produce probabilities over the vocabulary. The token with the highest probability is usually taken as the prediction, especially for text generation.
  + This is especially common in language modelling tasks where the goal is to predict the next word in a sequence (think ChatGPT).

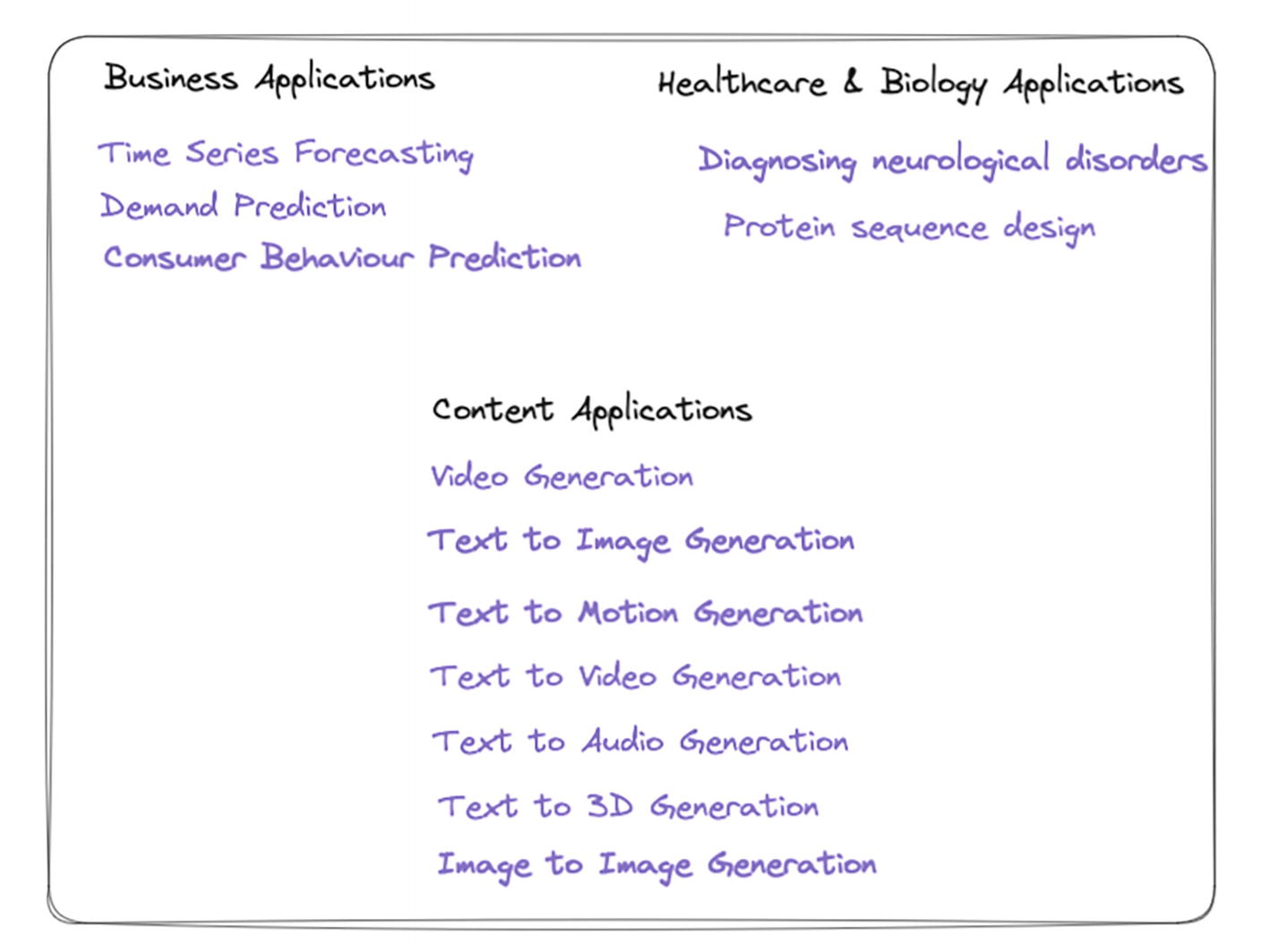
So far, you’ve learned about generative AI in the context of language, that is, large language models (LLMs); next up is diffusion models, which have gained popularity in the image generation space.

**Diffusion Explained**

Most recently, diffusion models have been used by the likes of OpenAI for DALL-E, Midjourney, and Stability AI, all for image generation. The way diffusion models work overall is actually quite simple – one of the less complex concepts we’ll discuss in this book.

Diffusion models are a type of generative model, which is used in a range of situations. You might already be very familiar with diffusion models being used for image and video generation. These models have also started showing promise in other areas such as drug discovery!

In Figure [1-4](https://learning.oreilly.com/library/view/building-generative-ai-powered/9798868802058/html/610509_1_En_1_Chapter.xhtml#Fig4), you can see just all the places diffusion models fit in.



***Figure 1-4***

Diffusion model applications

We’ll focus on images for the purpose of this book. Let’s take a look at how these models work.

**The Core Idea**

Imagine a drop of ink spreading out in a glass of water. This process of diffusion, where particles move from regions of high concentration to low concentration, is a natural phenomenon. In diffusion models, a reverse process is used: it starts from a target data point (like an image) and gradually adds noise to it until it becomes a simple distribution. The magic is that this process can be reversed to generate new data samples.

**How Diffusion Models Work**

1. 1.

**Noise Addition Process (Forward Process) (shown in** **Figure** [**1-5**](https://learning.oreilly.com/library/view/building-generative-ai-powered/9798868802058/html/610509_1_En_1_Chapter.xhtml#Fig5)**)**

* + Starts with a real data sample (e.g., a real image).
  + Gradually, it adds noise over several steps until the sample becomes indistinguishable from pure noise.

1. 2.

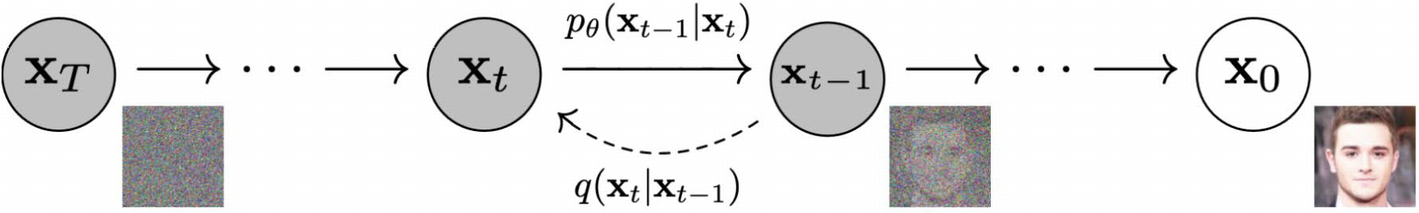
**Noise Removal Process (Reverse Process/Generation) (shown in Figure** [**1-5**](https://learning.oreilly.com/library/view/building-generative-ai-powered/9798868802058/html/610509_1_En_1_Chapter.xhtml#Fig5)**)**

* + Starts with a sample from a simple distribution (like Gaussian noise)
  + Uses a neural network to gradually remove the noise over several steps, guiding the sample to resemble a real data point from the target distribution

1. 3.

**Training**

* + During training, the model learns to reverse the noise addition process. It gets better at transforming a noisy sample into a realistic one.
  + This is done by using a neural network that predicts how to denoise a sample at each step. The model is trained on pairs of noisy samples and their less noisy versions.



***Figure 1-5***

Diffusion model process Image source: [https://​arxiv.​org/​pdf/​2006.​11239.​pdf](https://arxiv.org/pdf/2006.11239.pdf)

And there you have it, you now know how Stability AI, DALL-E, etc., work under the hood.

**What’s Next?**

So far you’ve learned about the architecture of the newly dubbed “foundation models.” As you read through this book, the next topics focused on are LangChain (your Swiss Army knife to AI apps), monitoring (can you really go into production without solid monitoring?), and finally fine-tuning these models. As AI summer progresses, it’s likely, rather than building and training models from scratch, you’ll be fine-tuning foundational models to your needs.

**Summary**

This chapter has given you an understanding of the two most popular architectures out there: transformers and diffusion models. These are foundational models that will form the basis of the AI applications you build in future. Both open source and closed source models make use of these architectures. As an AI engineer (practicing or aspiring), it’s important to understand what’s going on under the hood of the tools you use.