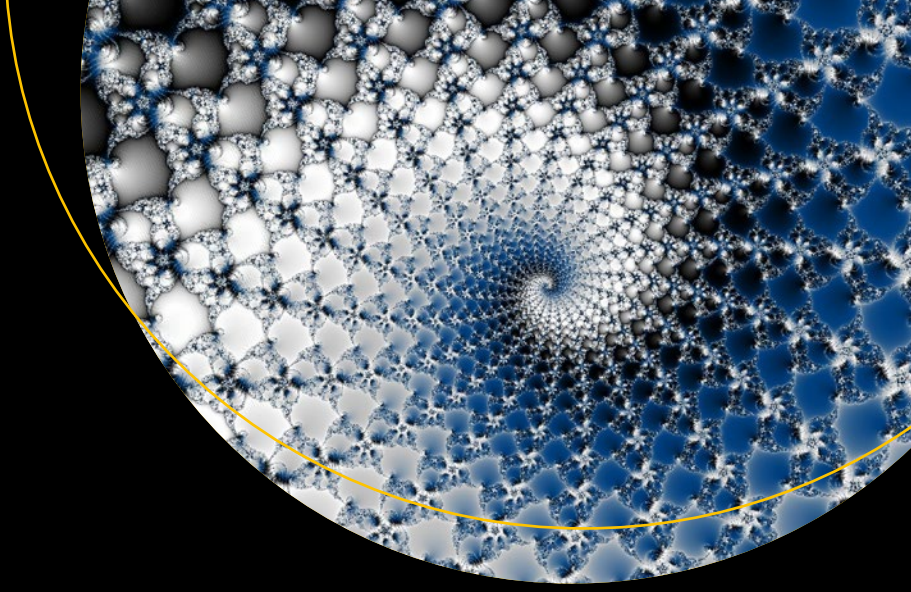

Building Generative AI-Powered Apps

A Hands-on Guide for Developers

Aarushi Kansal

Apress®

Building Generative AI-Powered Apps

A Hands-on Guide for Developers

Aarushi Kansal

Apress®

Building Generative AI-Powered Apps: A Hands-on Guide for Developers

Aarushi Kansal
Melbourne, Australia

ISBN-13 (pbk): 979-8-8688-0204-1 ISBN-13 (electronic): 979-8-8688-0205-8
https://doi.org/10.1007/979-8-8688-0205-8

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: Laura Berendson
Editorial Assistant: Gryffin Winkler

Cover designed by eStudioCalamar

Cover image designed by imagii on pixabay (pixabay.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, Suite 4600, New York, NY 10004-1562, USA. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub. For more detailed information, please visit https://www.apress.com/gp/services/source-code.

Paper in this product is recyclable

To my parents, for always backing my ambitious projects, even when they seemed a bit out there. This book is one of those dreams realized.

Table of Contents

About the Author

Aarushi Kansal is an experienced principal engineer. She has worked in a variety of technologies, including mobile development, Python, Go, and cloud, along with booming generative AI space. She has spearheaded AI initiatives in the workplace and regularly works on creative POCs in her spare time to stay at the top of the generative AI space.

About the Technical Reviewer

Akshay Kulkarni is an AI and machine learning evangelist and a thought leader. He has consulted several Fortune 500 and global enterprises to drive AI and data science–led strategic transformations. He is a Google Developer Expert, author, and regular speaker at major AI and data science conferences (including Strata, O'Reilly AI Conf, and GIDS). He is a visiting faculty member at some of the top graduate institutes in India. In 2019, he was also featured as one of the top 40 under 40 Data Scientists in India. In his spare time, he enjoys reading, writing, coding, and building next-gen AI products.

CHAPTER 1

Introduction to Generative AI

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.).

A. Kansal, *Building Generative AI-Powered Apps*,
https://doi.org/10.1007/979-8-8688-0205-8_1

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.

1. **Input**: At each step, the RNN takes in an input (e.g., a word in a sentence). **(Xn)**
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.
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**.
4. **Move to Next Step**: The process repeats for the next element in the sequence.

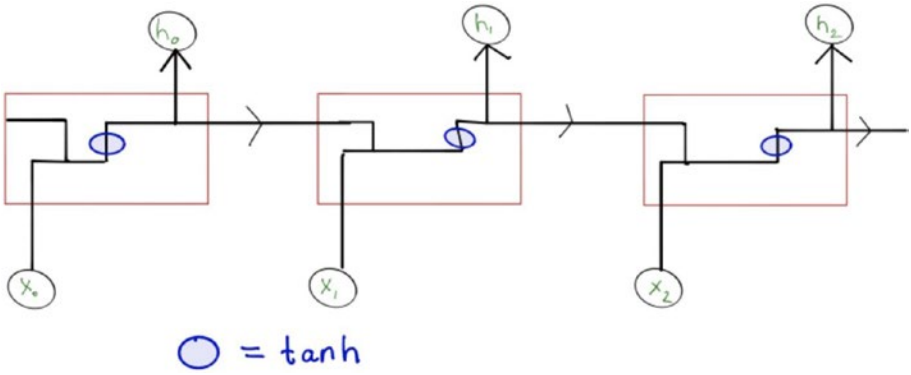

Figure 1-1. *RNN architecture*

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 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.

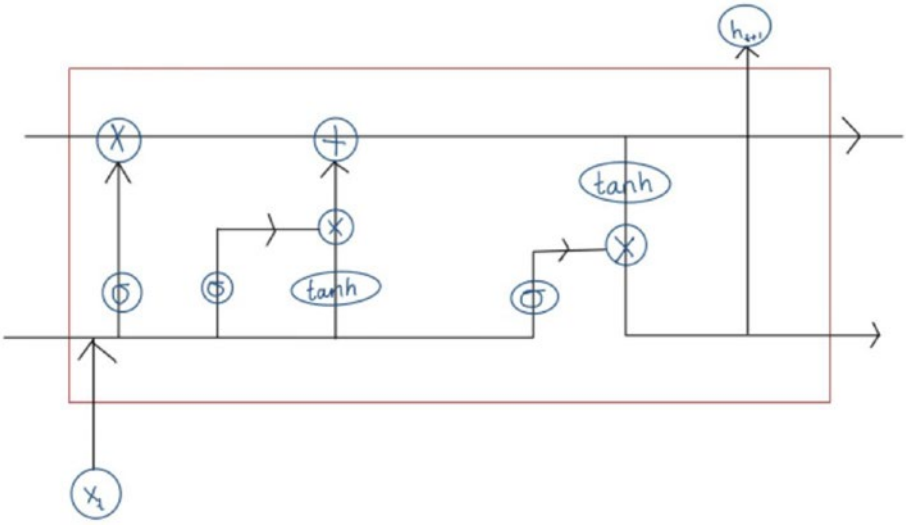

Figure 1-2. *LSTM cell*

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. **Input Vector:** Similar to RNNs, the LSTM unit takes in an input vector and the previous hidden state at each time step.
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.
3. **Output:** The LSTM produces an output, which is the hidden state passed to the next LSTM unit in the sequence.

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.