1ST EDITION

Coding with ChatGPT and Other LLMs

Navigate LLMs for effective coding, debugging, and AI-driven development

DR. VINCENT AUSTIN HALL

Coding with ChatGPT and Other LLMs

Navigate LLMs for effective coding, debugging, and AI-driven development

Dr. Vincent Austin Hall

‹packt›

Coding with ChatGPT and Other LLMs

The author acknowledges the use of cutting-edge AI, such as ChatGPT, with the sole aim of enhancing the language and clarity within the book, thereby ensuring a smooth reading experience for readers. It's important to note that the content itself has been crafted by the author and edited by a professional publishing team.

Group Product Manager: Niranjan Naikwadi

Publishing Product Manager: Nitin Nainani

Book Project Manager: Aparna Nair

Senior Editor: Joseph Sunil

Technical Editor: Rahul Limbachiya

Copy Editor: Safis Editing

Proofreader: Joseph Sunil

Indexer: Manju Arasan

Production Designer: Joshua Misquitta

Senior DevRel Marketing Executive: Vinishka Kalra

First published: November 2024

Production reference: 1061124

Published by Packt Publishing Ltd.

Grosvenor House

11 St Paul's Square

Birmingham

B3 1RB, UK.

ISBN 978-1-80512-505-1

www.packtpub.com

Contributors

About the author

Dr. Vincent Austin Hall is a computer science lecturer at Birmingham Newman University and CEO of Build Intellect Ltd, an AI consultancy. Build Intellect works closely with ABT News LTD, based in Reading, England. He holds a physics degree from the University of Leeds, an MSc in biology, chemistry, maths, and coding from Warwick, and a PhD in machine learning and chemistry, also from Warwick, where he developed licensed software for pharma applications. With experience in tech firms and academia, he's worked on ML projects in the automotive and medtech sectors. He supervises dissertations at the University of Exeter, consults on AI strategies, coaches students and professionals, and shares insights through blogs and YouTube content.

I would like to thank my supportive and patient family: my excellent and wise partner Anna, our brilliant, different, and loving son Peter and our brilliant, inventive, and hilarious daughter Lara, for allowing me time to work on this book over many weekends and evenings and understanding that good things take long, hard work, and many iterations.

Thank you to Packt Publishing: Editor Joseph Sunil for making only good suggestions and improving my work; Book Project Manager, Aparna Nair for keeping the project progressing well and making sure everything got done; Publishing Product Manager, Nitin Nainani for managing and further direction; Priyanshi J for bringing me on board and suggesting this book in the first place; as well as the technical reviewers for helping Joseph and me to keep the book quality high.

Thanks to my business partner, Chief Chigbo Uzokwelu, CEO of ABT News Ltd, for lots of support in friendship and business: legal, sales, business communications, proof reading, and marketing.

Thanks to the reader for reading and learning, sharing what you've learned and helping others to upskill and create the best code, careers and solutions for Earth (and future populated worlds).

About the reviewers

Parth Santpurkar is a senior software engineer with over a decade of industry experience based out of the San Francisco Bay area. He's a senior IEEE member and his expertise and interests range from software engineering and distributed systems to machine learning and artificial intelligence.

Sougata Pal is a passionate technology specialist performing the role of an enterprise architect in software architecture design and application scalability management, team building, and management. With over 15 years of experience, they have worked with different start-ups and large-scale enterprises to develop their business application infrastructure, enhancing their reach to customers. They have contributed to different open source projects on GitHub to empower the open source community. For the last couple of years, they have playing around with federated learning and cybersecurity algorithms to enhance the performance of cybersecurity processes by introducing concepts of federated learning.

Table of Contents

Part 3: Explainability, Shareability, and the Future of LLM-Powered Coding

8

9

10

Part 4: Maximizing Your Potential with LLMs: Beyond the Basics

11

Preface

In this age of the AI Revolution, you cannot achieve goals entirely with human power.

Automation is thousands of times faster and accelerating extremely quickly! Software ate the world and created AI. Now AI is eating the world and recreating it better. The best way to create is a fusion of human and machine powers.

In *Coding with ChatGPT and Other LLMs*, you will learn how coding is best achieved today. You can learn how to find and effectively use the most advanced tools for code generation, architecting, description and testing while staying out of legal hassles, advancing your career faster and helping others around you to improve too. After reading this book, its prompts and its code, you should understand likely futures for this kind of technology. You'll also be able to generate your own ideas about how to improve the world, and have the power to do that.

Who this book is for

This book is for new coders and experienced coders, software engineers, software developers, scientists doing scientific computing. If you want a career in coding or software, this book is for you. The book helps with ethics, bias, security, or the future impacts of AI, this book is for you.

If you are a lawyer concerned with the legal issues of AI and code, you'd do well to read this book.

What this book covers

Chapter 1, What is ChatGPT and What are LLMs?, introduces Large Language Models (LLMs) like ChatGPT and Claude. It explains how these models function and explores their applications through real-world examples.

Chapter 2, Unleashing the Power of LLMs for Coding: A Paradigm Shift, explores how LLMs can revolutionize software development by generating code. It introduces effective prompt strategies, highlights common pitfalls to avoid, and emphasizes the importance of iterative refinement for optimal results

Chapter 3, Code Refactoring, Debugging, and Optimization: A Practical Guide, delves into the essential tasks of refining code. It covers debugging to ensure functionality, refactoring to improve structure or adapt functionality, and optimizing for speed, memory usage, and code quality. The chapter demonstrates how LLMs can assist in these processes, providing practical strategies for effective AI-powered coding.

Chapter 4, Demystifying Generated Code for Readability, emphasizes the importance of writing clear, understandable code. It highlights how code that makes sense to its author may not be easily grasped by others—or even by the author at a later time. This chapter demonstrates how LLMs can help improve code readability by enhancing documentation, clarifying functions and libraries, and fostering practices that make the codebase more accessible for collaborators and your future self.

Chapter 5, Addressing Bias and Ethical Concerns in LLM-Generated Code, explores how biases can arise from the data used to train LLMs, implicit assumptions in prompts, or developer expectations. It provides strategies to identify hidden biases and correct them to ensure fair and responsible code generation.

Chapter 6, Navigating the Legal Landscape of LLM-Generated Code, discusses potential legal challenges related to biases, code reuse, copyright issues, and varying regulations across jurisdictions. This chapter equips you with the knowledge needed to address legal risks and ensure compliance when using LLM-generated code.

Chapter 7, Security Considerations and Measures, focuses on safeguarding your software from vulnerabilities. It highlights security risks that may emerge in LLM-generated code and provides best practices for identifying, mitigating, and preventing potential threats.

Chapter 8, Limitations of Coding with LLMs, addresses the boundaries of what LLMs can achieve. It explores their challenges in grasping the subtleties of human language and their limitations in handling complex coding tasks. The chapter also examines the inconsistencies and unpredictabilities inherent in LLM-generated outputs, helping readers set realistic expectations.

Chapter 9, Cultivating Collaboration in LLM-Enhanced Coding, promotes a culture of openness and collaboration in software development. It offers best practices for sharing code generated by LLMs and the knowledge that accompanies it, fostering transparency and teamwork. Readers will discover strategies to ensure the expertise encoded within LLM-generated solutions is effectively shared and utilized across development teams.

Chapter 10, Expanding the LLM Toolkit for Coders: Beyond LLMs, explores how non-LLM AI tools can complement LLM-powered coding. It highlights tools for code writing, analysis, and testing, detailing their capabilities and limitations. This chapter provides strategies for integrating these tools into a well-rounded coding toolkit to enhance productivity and maximize efficiency.

Chapter 11, Helping Others and Maximizing Your Career with LLMs, focuses on contributing to the LLM coding community through teaching, mentoring, and knowledge-sharing. It offers guidance on how to advance the field by sharing expertise and explores ways to leverage LLM-generated coding skills for career growth and new opportunities.

Chapter 12, The Future of LLMs in Software Development, looks ahead to emerging trends and developments in LLM technology. It reflects on how these advancements will shape the future of software development and examines the broader impact of automated coding on society, including potential implications for future communities.

To get the most out of this book

Assumed knowledge: some basic coding skills, an interest in software and or AI.

Try to apply what you've learned here, and share your code and your recent learnings and experience with others and learn from them.

Software/hardware covered in the book	Operating system requirements
Python	Windows, macOS, or Linux
Java	
HTML	
JavaScript	

Download the example code files

You can download the example code files for this book from GitHub at `https://github.com/PacktPublishing/Coding-with-ChatGPT-and-Other-LLMs`. If there's an update to the code, it will be updated in the GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Conventions used

There are a number of text conventions used throughout this book.

`Code in text:` Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and X handles. Here is an example: "Next, we have *Prompt 5* as a Flask app (`app.py`) with Python code."

A block of code is set as follows:

```
<!DOCTYPE html>
<html>
<head>
<title>Button Click</title>
<script>
function sayHello() {
  alert("Hello!");
}
</script>
</head>
<body>
<button onclick="sayHello()">Click me</button>
```

```
</body>
</html>
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
import pandas as pd
import matplotlib.pyplot as plt
# Sample data (replace with your data)
data = pd.Series([1, 2, 3, 4, 5])
# Assuming the data is in a column named "values"
fig, ax = plt.subplots()
ax.plot(data)
ax.set_xlabel("Index")
ax.set_ylabel("Value")
ax.set_title("Line Plot of Data")
plt.show()
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For instance, words in menus or dialog boxes appear in **bold**. Here is an example: "You'd have to click the first button, **Click me**, to get the pop-up window again."

> **Tips or important notes**
> Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, email us at customercare@packtpub.com and mention the book title in the subject of your message.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata` and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit authors.packtpub.com.

Share your thoughts

Once you've read *Coding with ChatGPT and Other LLMs*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below

`https://packt.link/free-ebook/978-1-80512-505-1`

2. Submit your proof of purchase
3. That's it! We'll send your free PDF and other benefits to your email directly

Part 1: Introduction to LLMs and Their Applications

This section lays the groundwork for understanding Large Language Models (LLMs) and their transformative potential across various fields. It introduces LLMs like ChatGPT, explaining how they work. We will also explore different ways that LLMs are applied across industries, from customer service to content generation. We will also check out the unique capabilities of LLMs in software development.

This section covers the following chapters:

- *Chapter 1, What is ChatGPT and what are LLMs?*
- *Chapter 2, Unleashing the Power of LLMs for Coding: A Paradigm Shift*
- *Chapter 3, Code Refactoring, Debugging, and Optimization: A Practical Guide*

1

What is ChatGPT and What are LLMs?

The world has been strongly influenced by the recent advancements in AI, especially **large language models** (**LLMs**) such as ChatGPT and Gemini (formerly Bard). We've witnessed stories such as OpenAI reaching one million users in five days, huge tech company lay-offs, history-revising image scandals, more tech companies getting multi-trillion dollar valuations (Microsoft and NVIDIA), a call for funding of $5–7 trillion for the next stage of technology, and talks of revolutions in how *everything* is done!

Yes, these are all because of new AI technologies, especially LLM tech.

LLMs are large in multiple ways: not just large training sets and large training costs but also large impacts on the world!

This book is about harnessing that power effectively, for your benefit, if you are a coder.

Coding has changed, and we must all keep up or else our skills will become redundant or outdated. In this book are tools needed by coders to quickly generate code and do it well, to comment, debug, document, and stay ethical and on the right side of the law.

If you're a programmer or coder, this is for you. Software, especially AI/machine learning, is changing everything at ever-accelerating rates, so you'll have to learn this stuff quickly, and then use it to create and understand future technologies.

I don't want to delay you any longer, so let's get into the first chapter.

In this chapter, we'll cover some basics of ChatGPT, Gemini, and other LLMs, where they come from, who develops them, and what the architectures entail. We'll introduce some organizations that use LLMs and their services. We'll also briefly touch on some mathematics that go into LLMs. Lastly, we'll check out some of the competition and applications of LLMs in the field.

This chapter covers the following topics:

- Introduction to LLMs
- Origins of LLMs
- Early LLMs
- Exploring modern LLMs
- How transformers work
- Applications of LLMs

Introduction to LLMs

ChatGPT is an LLM. LLMs can be used to answer questions and generate emails, marketing materials, blogs, video scripts, code, and even books that look a lot like they've been written by humans. However, you probably want to know about the technology.

Let's start with what an LLM is.

LLMs are deep learning models, specifically, transformer networks or just "*transformers*." Transformers certainly have transformed our culture!

An LLM is trained on huge amounts of text data, petabytes (thousands of terabytes) of data, and predicts the next word or words. Due to the way LLMs operate, they are not perfect at outputting text; they can give alternative facts, facts that are "hallucinated."

ChatGPT is, as of the time of writing, the most popular and famous LLM, created and managed by OpenAI. OpenAI is a charity and a capped-profit organization based in San Francisco [*OpenAI_LP*, *OpenAIStructure*].

ChatGPT is now widely used for multiple purposes by a huge number of people around the world. Of course, there's GPT-4 and now GPT-4 Turbo, which are paid, more powerful, and do more things, as well as taking more text in prompts.

It's called ChatGPT: *Chat* because that's what you do with it, it's a chatbot, and **GPT** is the technology and stands for **generative pre-trained transformer**. We will get more into that in the *GPT lineage* subsection.

A transformer is a type of neural network architecture, and a transformer is the basis of the most successful LLMs today (2024). GPT is a Generative Pre-trained Transformer. Gemini is a transformer [*ChatGPT*, *Gemini*, *Menon*, *HuggingFace*]. OpenAI's GPT-4 is a remarkable advancement in the field of AI. This model, which is the fourth iteration of the GPT series, has introduced a new feature: the ability to generate images alongside text. This is a significant leap from its predecessors, which were primarily text-based models.

OpenAI also has an image generation AI, DALL-E, and an AI that can connect images and text and does image recognition, called CLIP (*OpenAI_CLIP*). The image generation capability of DALL-E is achieved by training the transformer model on image data. This means that the model has been exposed to a vast array of images during its training phase, enabling it to understand and generate visual content [*OpenAI_DALL.E*].

Furthermore, since images can be sequenced to form videos, DALL.E can also be considered a video generator. This opens up a plethora of possibilities for content creation, ranging from static images to dynamic videos. It's a testament to the versatility and power of transformer models, and a glimpse into the future of AI capabilities.

In essence, tools from OpenAI are not just text generators but a comprehensive suite of content generators, capable of producing a diverse range of outputs. It's called being **multi-modal**. This makes these tools invaluable in numerous applications, from content creation and graphic design to research and development. The evolution from GPT-3 to GPT-4 signifies a major milestone in AI development, pushing the boundaries of what AI models can achieve.

Origins of LLMs

Earlier neural networks with their ability to read sentences and predict the next word could only read one word at a time and were called **recurrent neural networks**, (**RNNs**). RNNs attempted to mimic human-like sequential processing of words and sentences but faced challenges in handling long-term dependencies between words and sentences due to very limited memory capacity.

In 1925, the groundwork was laid by Wilhelm Lenz and Ernst Ising with their non-learning Ising model, considered an early RNN architecture [*Brush*, *Gemini*].

In 1972, Shun'ichi Amari made this architecture adaptive, paving the way for learning RNNs. This work was later popularized by John Hopfield in 1982 [*Amari*, *Gemini*].

Due to this, there has been a fair amount of research to find ways to stretch this memory to include more text to get more context. RNNs are transformers. There are other transformers, including **LSTMs**, which are **long short-term memory** neural networks that are based on a more advanced version of RNNs, but we won't go into that here [*Brownlee_LLMs*, *Gemini*]. LSTMs were invented by Hochreiter and Schmidhuber in 1997 [*Wiki_LSTM*, *Hochreiter1997*].

There is another network called the **convolutional neural network** (**CNN**). Without going into much detail, CNNs are very good at images and lead the world in image recognition and similar jobs. CNNs (or ConvNets) were invented in 1980 by Kunihiko Fukushima and developed by Yann LeCun, but they only really became popular in the 2000s, when GPUs became available. Chellapilla *et al*. tested the speeds of training CNNs on CPUs and GPUs and found the network trained on GPUs 4.1 times faster [*Fukushima1980*, *LeCun1989*, *Chellapilla2006*]. Sometimes, your inventions take time to bear fruit, but keep inventing! CNNs use many layers or stages to do many different mathematical things to their inputs and try to look at them in different ways: different angles, with detail taken out (dropout layers), pooling nearby regions of each image, zeroing negative numbers, and other tricks.

What was needed was a model with some form of memory to remember and also generate sentences and longer pieces of writing.

In 2017, Ashish Vaswani and others published a paper called *Attention Is All You Need*, [*Vaswani*, *2017*]. In this important paper, the transformer architecture was proposed based on attention mechanisms. In other words, this model didn't use recurrence and convolutions, such as RNNs and CNNs. These methods have been very successful and popular AI architectures in their own right.

Compared to RNNs and CNNs, Vaswani's Transformer performed faster training and allowed for higher parallelizability.

The Transformer was the benchmark for English-to-German translation and established a new state-of-the-art single model in the WMT 2014 English-to-French translation task. It also performed this feat after being trained for a small fraction of the training times of the next best existing models. Indeed, Transformers were a groundbreaking advancement in natural language processing [*Vaswani*, *2017*].

Now that we have covered the origins of LLMs, we will check out some of the earliest LLMs that were created.

Early LLMs

There are many LLMs today and they can be put into a family tree; see *Figure 1.1*. The figure shows the evolution from word2vec to the most advanced LLMs in 2023: GPT-4 and Gemini [*Bard*].

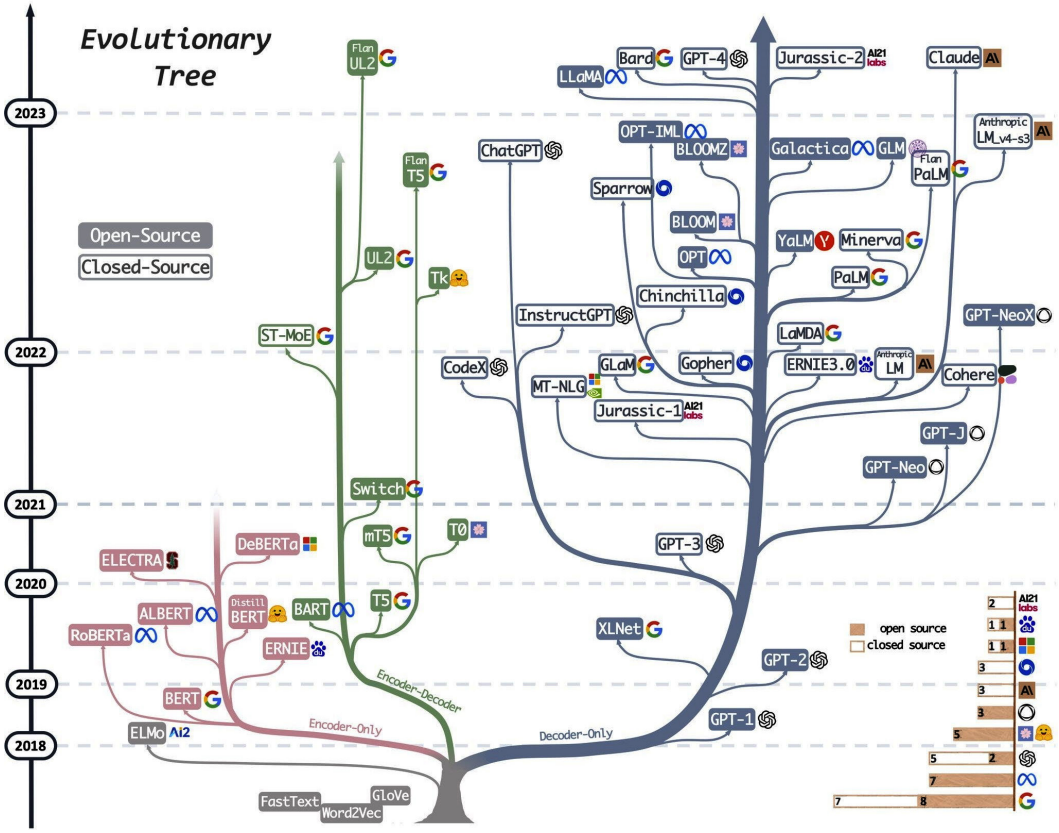

Figure 1.1: Family tree of LLMs from word2vec to GPT-4 and Bard, from Yang2023 with permission

So, that's all of them but, for now, we'll look at the earlier LLMs that lead to the most advanced technologies today. We'll start with GPT.

GPT lineage

The development of GPT is a constantly changing and iterative process, with each new model building upon the strengths and weaknesses of its ancestors. The GPT series, initiated by OpenAI, has undergone a great deal of evolution, leading to advancements in **natural language processing** (**NLP**) and understanding.

GPT-3, the third iteration, brought a significant leap in terms of size and complexity, with an impressive 175 billion parameters. This allowed it to generate pretty human-like text across a wide range of topics and subjects [*Wiki_GPT3*, *ProjectPro*].

As the GPT series progressed, OpenAI continued to refine and enhance the architecture. In subsequent iterations, GPT-4 and GPT-4 Turbo have further pushed back the boundaries of what these LLMs can achieve. The iterative development process focuses on increasing model size and improving fine-tuning capabilities, enabling more nuanced and contextually relevant outputs.

Further to this, there are more modalities, such as GPT-4 with vision and text-to-speech.

GPT model iteration is not solely about scaling up the number of parameters; it also involves addressing the limitations observed in earlier versions. Feedback from user interactions, research findings, and technological advancements contribute to the iterative nature of the GPT series. OpenAI is constantly working to reduce the amount of inaccurate information and incoherent outputs (hallucinations) that its chatbots produce. Also, each iteration of the chatbot takes on board the lessons learned from real-world applications and user feedback.

GPT models are trained and fine-tuned on very large, diverse datasets to make sure the chatbots can adapt to many different contexts, industries, and user requirements. The iterative development approach ensures that later GPT models are better equipped to understand and generate human-like text, making them extremely valuable tools for a huge number of applications, including content creation such as blogs, scripts for videos, and copywriting (writing the text in adverts) as well as conversational agents (chatbots and AI assistants).

The way GPT models are developed iteratively shows OpenAI's commitment to continuous improvement and innovation in the field of LLMs, allowing even more sophisticated and capable models to be built from these models in the future.

Here are the dates for when the different versions of GPT were launched:

- GPT was first launched in June 2018
- GPT-2 was released in February 2019
- GPT-3 in 2020
- GPT-3.5 in 2022/ChatGPT in November 2022

There will be more on the GPT family later, in the *GPT-4 /GPT-4 Turbo* section.

Here, we will detail the architecture of LLMs and how they operate.

BERT

To comprehend the roots and development of **Bidirectional Encoder Representations from Transformers** (**BERT**), we must know more about the intricate and fast-moving landscape of neural networks. Without hyperbole, BERT was a seriously important innovation in NLP, part of the ongoing evolution of AI. BERT was the state of the art for a wide range of NLP tasks in October 2018, when it was released [*Gemini*]. This included question answering, sentiment analysis, and text summarization.

BERT also paved the way for later R&D of LLMs; it played a pivotal role in LLM development. BERT, being open source, helped to speed up LLM advancement.

BERT takes some of its DNA from RNNs (mentioned in the *Origins of LLMs* section), the neural nets that loop back on themselves to create a kind of memory, although rather limited memory.

The invention of the first transformer architecture was key to the origin of BERT. The creation of BERT as a bidirectional encoder (these go backward and forward along a sentence) drew inspiration from the transformer's attention-based mechanism, allowing it to capture contextual relationships between words in both directions within a sentence.

So, BERT's attention is bidirectional (left-to-right and right-to-left context). At its creation, this was unique, and it enabled BERT to gain a more comprehensive understanding of nuanced language semantics.

While BERT's foundations are in transformer architecture, its characteristics have evolved with further research and development, though it is not currently in development. Each iteration of BERT refined and expanded its capabilities.

The BERT LLM was a stage of the ongoing innovation in AI. BERT's ability to understand language bidirectionally, drawing insights from both preceding and succeeding words, is part of the endeavors taken to achieve the creation of an AI with a sufficiently deep awareness of the intricacies of natural language.

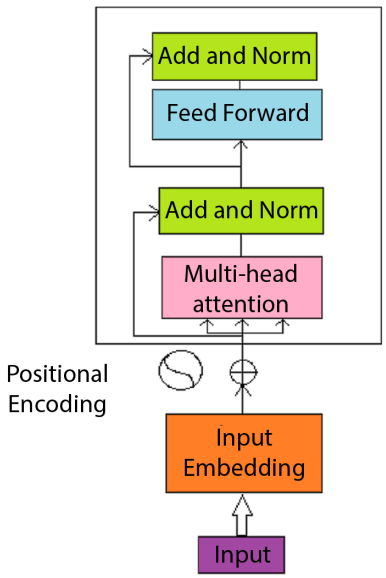

Figure 1.2: Architecture of BERT, a bidirectional encoder (reproduced from GeekCultureBERT)

LaMDA

Understanding the ancestry of **Language Model for Dialogue Applications** (**LaMDA**) involves tracing the roots of its architectural design and the evolutionary path it followed in the landscape of NLP. LaMDA, like its counterparts, emerges from a family of models that have collectively revolutionized how machines comprehend and generate human-like text.

RNNs, mentioned in this chapter's first section, play a pivotal role in LaMDA's family tree.

The breakthrough came with the invention of transformer architectures, and LaMDA owes a significant debt to the transformative *Attention Is All You Need* paper [*Vaswani 2017*, *2023*]. This paper laid the groundwork for a novel approach, moving away from sequential processing to a more parallelized and attention-based mechanism.

The LaMDA LLM inherits its core architecture from the transformer family and was developed by Google. These models learn very well how words in a sentence relate to each other. This allows a transformer to have a richer understanding of language. This change from using traditional processing in sequence was a paradigm shift in NLP, enabling LaMDA to more effectively grasp nuanced interactions and dependencies within texts.

While the origins lie in the transformer architecture, LaMDA's unique characteristics may have been fine-tuned and evolved through subsequent research and development efforts. LaMDA's lineage is not just a linear progression but a family tree, a branching exploration of many possibilities, with each iteration refining and expanding its capabilities. In *Figure 1.1*, LaMDA is near ERNIE 3.0, Gopher, and PaLM on the right of the main, vertical blue branch.

Simply put, LaMDA is a product of ongoing innovation and refinement in the field of AI, standing on the shoulders of earlier models and research breakthroughs. Its ability to comprehend and generate language is deeply rooted in an evolutionary process of learning from vast amounts of text data, mimicking the way humans process and understand language on a grand, digital scale.

LaMDA was launched in May 2021.

LLaMA's family tree

LLaMA is the AI brainchild of Meta AI. It might not be one you've heard the most about but its lineage holds stories of innovation and evolution, tracing a fascinating path through the history of AI communication.

Like the other chatbot LLMs, LLaMA's roots are also in transformer architectures. These models rely on intricate attention mechanisms, allowing them to analyze relationships between words, not just their sequence.

Trained on massive datasets of text and code, LLaMA learned to generate basic responses, translate languages, and even write different kinds of creative text formats.

However, like a newborn foal, their capabilities were limited. They stumbled with complex contexts, lacked common sense reasoning, and sometimes sputtered out nonsensical strings.

Yet their potential was undeniable. The ability to learn and adapt from data made them valuable tools for researchers. Meta AI nurtured these nascent models, carefully tweaking their architecture and feeding them richer datasets. They delved deeper into the understanding of human language, acquiring skills such as factual grounding, reasoning, and the ability to engage in multi-turn conversations (Wiki_llama).

The Llama family tree is not a linear progression but, rather, a family of multiple branches of exploration. Different versions explored specific avenues: Code Llama focused on code generation, while Megatron-Turing NLG 530 B was trained on filling in missing words, reading comprehension, and common-sense reasoning, among other things (*CodeLlama 2023*, *Megatron-Turing 2022*).

For an idea of how LLaMA fits into the evolutionary tree, see *Figure 1.1* at the top left of the vertical blue branch, near Bard (*Gemini*).

Each experiment, each successful leap forward, contributed valuable DNA to future generations.

Why the name *Megatron-Turing NLG 530 B*? *Megatron* because it represents a powerful hardware and software framework. *Turing* to honor Alan Turing, the first AI researcher, and the originator of AI and ML. **NLG** stands for **natural language generation**, and it has 530 billion parameters.

Meta AI continues to shepherd the Llama family, and the future promises more exciting developments.

Llama LLM was launched in February 2023, while Megatron-Turing NLG 530 B was released in January 2022.

Now that we have covered the origins and explored the early stages of LLMs, let us fast-forward and talk about modern LLMs in the next section.

Exploring modern LLMs

After the explosive take-off of ChatGPT in late 2022, with 1 million active users in 5 days and 100 million active users in January 2023 (about 2 months), 2023 was a pretty hot year for LLMs, AI research, and the use of AI in general.

Most tech companies have worked on their own LLMs or transformer models to use and make publicly available. Many companies, organizations, and individuals (students included) have used LLMs for a multitude of tasks. OpenAI keeps updating its GPT family and Google keeps updating its Bard version. Bard became Gemini in February 2024, so all references to Bard have changed to Gemini. Many companies use ChatGPT or GPT-4 as the core of their offering, just creating a wrapper and selling it.

This might change as OpenAI keeps adding modalities (speech, image, etc.) to the GPTs and even a new marketplace platform where users can create and sell their own GPT agents right on OpenAI servers. This was launched in early January 2024 to paid users ($20/month before VAT). We'll cover some of the latest LLMs that companies have worked on in the following sections.