# Introduction

This book will concentrate on the essential tech stack identified for adapting a large language model (LLM) to a specific use case and achieving a sufficient threshold of accuracy and reliability for scalable use by paying customers. Specifically, it will cover Prompt Engineering, Fine-tuning, and Retrieval-Augmented Generation (RAG).

Building your own production-ready apps and products using these models still requires a significant development effort. Hence, this book requires intermediate knowledge of Python. Although no programming knowledge is necessary to explore the AI and LLM-specific concepts in this book, we recommend using the list of useful and free Python resources for a more hands-on learning experience.

We are currently working on a course on Python for LLMs. In the meantime, the first few chapters of this book should still be light and easily understandable. In parallel, we would advise to take a look at Python and other resources we have to grow your AI technical skills and understanding. Going through one or two of the Python resources listed at towardsai.net/book should be enough to set you up for this book. Once you are more confident in your programming skills, return to code-centric sections.

Despite significant efforts by central AI labs and open-source developers in areas like Reinforcement Learning with Human Feedback to adapt foundation models to human requirements and use cases, off-the-shelf foundation models still have limitations that restrict their direct use in production, except for the most straightforward tasks.

There are various ways to adapt an off-the-shelf "foundation model" LLM to a specific application and use case. The initial decision is whether to use an LLM via API or a more flexible platform where you have full access to the model weights. Some may also want to experiment with training their own models; however, in our opinion, this will rarely be practical or economical outside the leading AI labs and tech companies. Over 5 million people are now building upon LLMs on platforms such as OpenAI, Anthropic, Nvidia, and Hugging Face. This book walks you through overcoming LLM's limitations and developing LLM products that are ready for production with key tech stacks!

---

## Why Prompt Engineering, Fine-Tuning, and RAG?

LLMs such as GPT-4 often lack domain-specific knowledge, making generating accurate or relevant responses in specialized fields challenging. They can also struggle with handling large data volumes, limiting their utility in data-intensive scenarios. Another critical limitation is their difficulty processing new or technical terms, leading to misunderstandings or incorrect information. Hallucinations, where LLMs produce false or misleading information, further complicate their use. Hallucinations are a direct result of the model training goal of the next token prediction - to some extent, they are a feature that allows "creative" model answers. However, it is difficult for an LLM to know when it is answering from memorized facts and imagination. This creates many errors in LLM-assisted workflows, making them difficult to identify. Alongside hallucinations, LLMs sometimes also simply fail to use available data effectively, leading to irrelevant or incorrect responses.

LLMs are generally used in production for performance and productivity-enhancing “copilot” use cases, with a human still fully in the loop rather than for fully automated tasks due to these limitations. But there is a long journey from a basic LLM prompt to sufficient accuracy, reliability, and observability for a target copilot use case. This journey is called the “march of 9s” and is popularized in self-driving car development. The term describes the gradual improvement in reliability, often measured in the number of nines (e.g., 99.9% reliability) needed to reach human-level performance eventually.

We think the key developer tool kit for the “march of 9s” for LLM-based products is 1) Prompt Engineering, 2) Retrieval-Augmented Generation (RAG), 3) Fine-Tuning, and 4) Custom UI/UX. In the near term, AI can assist many human tasks across various industries by combining LLMs, prompting, RAG, and fine-tuning workflows. We think the most successful “AI” companies will focus on highly tailored solutions for specific industries or niches and contribute a lot of industry-specific data and intelligence/experience to how the product is developed.

RAG consists of augmenting LLMs with specific data and requiring the model to use and source this data in its answer rather than relying on what it may or may not have memorized in its model weights. We love RAG because it helps with:

1) Reducing hallucinations by limiting the LLM to answer based on existing chosen data.
2) Helping with explainability, error checking, and copyright issues by clearly referencing its sources for each comment.
3) Giving private/specific or more up-to-date data to the LLM.
4) Not relying too much on black box LLM training/fine-tuning for what the models know and have memorized.

Another way to increase LLM performance is through good prompting. Multiple techniques have been found to improve model performance. These methods can be simple, such as giving detailed instructions to the models or breaking down big tasks into smaller ones to make them easier for the model to handle. Some prompting techniques are:

1) "Chain of Thought" prompting involves asking the model to think through a problem step by step before coming up with a final answer. The key idea is that each token in a language model has a limited "processing bandwidth" or "thinking capacity." The LLMs need these tokens to figure things out. By asking it to reason through a problem step by step, we use the model's total capacity to think and help it arrive at the correct answer.
2) "Few-Shot Prompting" is when we show the model examples of the answers we seek based on some given questions similar to those we expect the model to receive. It's like showing the model a pattern of how we want it to respond.
3) "Self-Consistency" involves asking the same question to multiple versions of the model and then choosing the answer that comes up most often. This method helps get more reliable answers.
In short, good prompting is about guiding the model with clear instructions, breaking down tasks into simpler ones, and using specific methods to improve performance. It's basically the same steps we must do when starting new assignments. The professor assumes you know the concepts and asks you to apply them intelligently.

On the other hand, fine-tuning is like giving the language model extra lessons to improve output for specific tasks. For example, if you want the model to turn regular sentences into SQL database queries, you can train it specifically on that task. Or, if you need the model to respond with answers in JSON format—a type of structured data used in programming—you can fine-tune it. This process can also help the model learn specific information about a certain field or subject. However, if you want to add specialized knowledge quickly and more efficiently, Retrieval-Augmented Generation (RAG) is usually a better first step. With RAG, you have more control over the information the model uses to generate responses, making the experimentation phase quicker, more transparent, and easier to manage.

Parts of this toolkit will be partially integrated into the next generation of foundation models, while parts will be solved through added frameworks like LlamaIndex and LangChain, especially for RAG workflows. However, the best solutions will need to tailor these tools to specific industries and applications. We also believe prompting, along with RAG, are here to stay - over time, prompting will resemble the necessary skills for effective communication and delegation to human colleagues. While it's there to stay, the libraries are constantly evolving. We have linked to the documentation of both LlamaIndex and LangChain on towardsai.net/book for the most up-to-date information.

The potential of this generation of AI models goes beyond typical natural language processing (NLP) tasks. There are countless use cases, such as explaining complex algorithms, building bots, helping with app development, and explaining academic concepts. Text-to-image programs like DALL-E, Stable Diffusion, and Midjourney revolutionize fields like animation, gaming, art, movies, and architecture. Additionally, generative AI models have shown transformative capabilities in complex software development with tools like GitHub Copilot.

---

## The Current LLM Landscape

The breakthroughs in Generative AI have left us with an extremely active and dynamic landscape of players. This consists of 1) AI hardware manufacturers such as Nvidia, 2) AI cloud platforms such as Azure, AWS, and Google, 3) Open-source platforms for accessing the full models, such as Hugging Face, 4) Access to LLM models via API such as OpenAI, Cohere and Anthropic and 5) Access to LLMs via consumer products such as ChatGPT, Perplexity and Bing. Additionally, many more breakthroughs are happening each week in the AI universe, like the release of multimodal models (that can understand both text and image), new model architectures (such as a Mixture of Experts), Agent Models (models that can set tasks and interact with each other and other tools), etc.

---

# Coding Environment and Packages

All the code notebooks, Google colabs, GitHub repos, research papers, documentation, and other resources are accessible at towardsai.net/book.

To follow the coding sections of this book, you need to ensure that you have the appropriate coding environment ready. Make sure to use a Python version equal to or later than **3.8.1**. You can set up your environment by choosing **one of the following options**:

1. Having a code editor installed on your computer. A popular coding environment is Visual Studio Code, which uses Python virtual environments to manage Python libraries.
2. Using our Google Colab notebooks.

Note: Depending on when you purchase the book, parts of the code in the notebooks and Google Colab notebooks might require some change. We will update the code as regularly as possible to make the most up-to-date version available.

## Run the code locally

If you choose the first option, you will need the following packages to execute the sample codes in each section successfully. You will also need an environment set up.

Python virtual environments offer an excellent solution for managing Python libraries and avoiding package conflicts. They create isolated environments for installing packages, ensuring that your packages and their dependencies are contained within that environment. This setup provides clean and isolated environments for your Python projects.

Execute the `python` command in your terminal to confirm that the Python version is either equal to or greater than 3.8.1. Then follow these steps to create a virtual environment:

1. Create a virtual environment using the command: `python -m venv my_venv_name`.
2. Activate the virtual environment: `source my_venv_name/bin/activate`.
3. Install the required libraries and run the code snippets from the lessons within the virtual environment.

They can be installed using the pip packages manager. A link to this requirements text file is accessible at towardsai.net/book.

```
deeplake==3.6.19
openai==0.27.8
tiktoken==0.4.0
transformers==4.32.0
torch==2.0.1
numpy==1.23.5
deepspeed==0.10.1
```

```
trl==0.7.1

peft==0.5.0

wandb==0.15.8

bitsandbytes==0.41.1

accelerate==0.22.0

tqdm==4.66.1

neural_compressor===2.2.1

onnx===1.14.1

pandas==2.0.3

scipy==1.11.2
```

While we strongly recommend installing the latest versions of these packages, please note that the codes have been tested with the versions specified in parentheses. Moreover, specific lessons may require the installation of additional packages, which will be explicitly mentioned. The following code will demonstrate how to install a package using pip:

```
pip install deeplake
# Or: (to install an specific version)
# pip install deeplake==3.6.5
```

## Google Colab

Google Colaboratory, popularly known as Google Colab, is a *free cloud-based Jupyter notebook environment*. Data scientists and engineers widely use it to train machine learning and deep learning models using CPUs, GPUs, and TPUs. Google Colab comes with an array of features, such as:

• Free access to GPUs and TPUs for accelerated model training.

• A web-based interface for a service running on a virtual machine, eliminating the need for local software installation.
• Seamless integration with Google Drive and GitHub.
You need only a Google account to use Google Colab. You can run terminal commands directly in notebook cells by appending an exclamation mark (!) before the command. Every notebook created in Google Colab is stored in your Google Drive for easy access.

A convenient way of using API keys in Colab involves:

1. Saving the API keys in a file named `.env` on your Google Drive. Here's how the file should be formatted to save the Activeloop token and the OpenAI API key:

```
OPENAI_API_KEY=your_openai_key
```

1. Mounting your Google Drive on your Colab instance.
1. Loading them as environment variables using the `dotenv` library:

```
from dotenv import load_dotenv


load_dotenv('/content/drive/MyDrive/path/to/.env')
```

---

# Learning Resources

To help you with your learning process, we are sharing our open-source AI Tutor chatbot (aitutor.towardsai.net) to assist you when needed. This tool has been created using the same tools we teach in this book. We build a RAG system that provides an LLM with access to the latest documentation from all significant tools, such as LangChain and LlamaIndex, including our previous free courses. If you have any questions or require help during your AI learning journey, whether as a beginner or an expert in the field, you can reach out to our community members and the writers of this book in the dedicated channel (space) for this book in our Learn AI Together Discord Community: discord.gg/learnaitogether.

💡Several additional resources shared throughout the book are accessible at towardsai.net/book.