FROM SCRATCH

BUILD A Reasoning Model

Sebastian Raschka

MEAP

MANNING

MEAP Edition
Manning Early Access Program

Build a Reasoning Model (From Scratch)

Version 1

welcome

Thank you for purchasing the MEAP for *Build a Reasoning Model (From Scratch)*.

If you are like most people these days, LLMs are already part of your everyday toolkit. Maybe you have asked an LLM to proofread an email, debug a tricky piece of code, or explain a concept that sent you down an unexpected rabbit hole. Since 2022, and the launch of ChatGPT, these models have moved from experimental novelties to essential tools in our daily work and learning.

It's been quite a journey to get here. The earliest GPT models, introduced in 2018, could generate text that was more or less human-like, but they were primarily text-completion models, largely unable to answer even simple queries, and the response quality was nowhere near that of the LLMs we use today.

Then came instruction fine-tuning and alignment with human preferences, which ChatGPT popularized in 2022. The techniques behind ChatGPT transformed LLMs into the everyday problem solvers we use today. Currently, we are in the latest phase: developing reasoning models. Reasoning is the ability for an LLM to tackle more complex problems step-by-step.

Reasoning is one of the most exciting and important recent advances in improving LLMs, but it's also one of the easiest to misunderstand if you only hear the term reasoning and read about it in theory. That's why this book takes a hands-on approach. We'll start with a pre-trained base LLM and then add reasoning capabilities ourselves, step by step in code, so you can see exactly how it works.

This book isn't a "production deployment" manual, and we won't use any third-party LLM libraries. Instead, think of it as a behind-the-scenes tour where you get to develop the machinery yourself.

By the end, you will not only understand what reasoning is and how it works, but you will also have built it from scratch. That's a perspective that will serve you well whether you are using, developing, or planning to deploy LLMs in the future.

— Sebastian Raschka, PhD

brief contents

1 Understanding reasoning models

This chapter covers

- **What "reasoning" means specifically in the context of LLMs**
- **How reasoning differs from pattern matching**
- **The conventional pre-training and post-training stages of LLMs**
- **Key approaches to improving reasoning abilities in LLMs**
- **Why building reasoning models from scratch can improve our understanding of their strengths, limitations, and practical trade-offs**

Welcome to the next stage of large language models (LLMs): *reasoning*. LLMs have transformed how we process and generate text, but their success has been largely driven by statistical pattern recognition. However, new advances in reasoning methodologies now enable LLMs to tackle more complex tasks, such as solving logical puzzles and advanced math problems involving multi-step arithmetic. Understanding these methodologies is the central focus of this book.

This book teaches the inner workings of those LLM reasoning methods through a hands-on, code-first approach. As the title says, you will be creating a reasoning LLM from scratch. By the end of the book, you will understand how reasoning models work and be equipped to design, prototype, and evaluate the main methods for improving reasoning in LLMs.

With its focus on practical applications and explanations, this book is written to speak to LLM engineers, machine learning researchers, applied scientists, and software developers alike. Those new to LLMs may benefit from my book Build a Large Language Model (From Scratch), also published by Manning (http://mng.bz/orYv).

This first chapter introduces the foundational concepts before the following chapters shift toward practical, hands-on coding examples to directly implement reasoning techniques for LLMs.

1.1 Defining reasoning in the context of LLMs

What is LLM-based reasoning? The answer and discussion of this question itself would provide enough content to fill a book. However, this would be a different kind of book than this practical, hands-on coding focused book that implements LLM reasoning methods from scratch rather than arguing about reasoning on a conceptual level. Nonetheless, I think it's important to briefly define what we mean by reasoning in the context of LLMs.

So, before we transition to the coding portions of this book in the upcoming chapters, I want to kick off this book with this section that defines reasoning in the context of LLMs, and briefly explain how it relates to pattern matching and logical reasoning. This will lay the groundwork for further discussions on how LLMs are currently built, how they handle reasoning tasks, and what they are good and not so good at.

This book's definition of reasoning, in the context of LLMs, is as follows:

> *Reasoning, in the context of LLMs, refers to the model's ability to produce intermediate steps before providing a final answer. This is a process that is often described as chain-of-thought (CoT) reasoning. In CoT reasoning, the LLM explicitly generates a structured sequence of statements or computations that illustrate how it arrives at its conclusion.*

Figure 1.1 illustrates a simple example of multi-step (CoT) reasoning in an LLM.

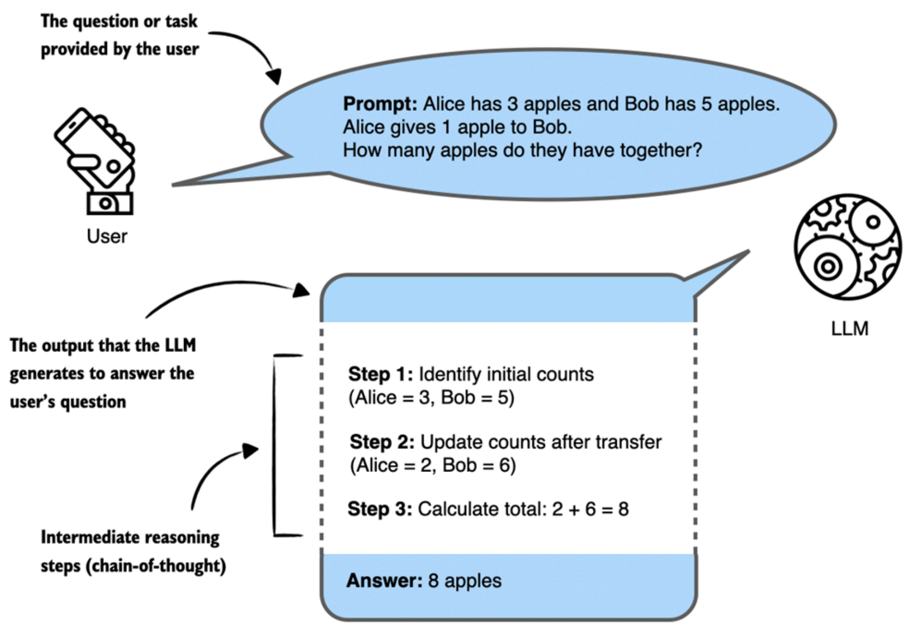

The question or task provided by the user
Prompt: Alice has 3 apples and Bob has 5 apples. Alice gives 1 apple to Bob. How many apples do they have together?
User
LLM
The output that the LLM generates to answer the user's question
Step 1: Identify initial counts (Alice = 3, Bob = 5)
Step 2: Update counts after transfer (Alice = 2, Bob = 6)
Step 3: Calculate total: 2 + 6 = 8
Intermediate reasoning steps (chain-of-thought)
Answer: 8 apples

Figure 1.1 A simplified illustration of how an LLM might tackle a multi-step reasoning task. Rather than just recalling a fact, the model needs to combine several intermediate reasoning steps to arrive at the correct conclusion. The intermediate reasoning steps may or may not be shown to the user, depending on the implementation.

LLM-produced intermediate reasoning steps, as shown in figure 1.1, look very much like a person articulating internal thoughts aloud. Yet how closely these methods (and the resulting reasoning processes) mirror human reasoning remains an open research question, one this book does not attempt to answer. It's not even clear that such a question can be definitively answered.

Instead, this book focuses on explaining and implementing the fundamental techniques that improve LLM-based reasoning and thus make LLMs better at handling complex tasks. My hope is that by gaining hands-on experience with these methods, you will be better prepared to understand and improve those reasoning methods being developed and maybe even explore how they compare to human reasoning.

LLM VERSUS HUMAN REASONING

Reasoning processes in LLMs may closely resemble human thought, particularly in how intermediate steps are articulated. However, it's not (yet) clear whether LLM reasoning mirrors human reasoning in terms of internal cognitive processes. Humans often reason by consciously manipulating concepts, intuitively understanding abstract relationships, or generalizing from a few examples. In contrast, current LLM reasoning is primarily based on patterns learned from extensive statistical associations present in training data, rather than explicit internal cognitive structures or conscious reflection as recently evidenced by a study from a research team at Apple, *The Illusion of Thinking: Understanding the Strengths and Limitations of Reasoning Models via the Lens of Problem Complexity* (https://machinelearning.apple.com/research/illusion-of-thinking).

In short, although the outputs of reasoning-enhanced LLMs can appear human-like, the underlying mechanisms (likely) differ substantially and remain an active area of exploration.

1.2 Understanding the standard LLM training pipeline

This section briefly summarizes how conventional (non-reasoning) LLMs are typically trained so that we can understand where their limitations lie. This background will also help frame our upcoming discussions on the differences between pattern matching and logical reasoning.

Before applying any specific reasoning methodology, conventional LLM training is usually structured into two stages: *pre-training* and *post-training*, which are illustrated in figure 1.2.

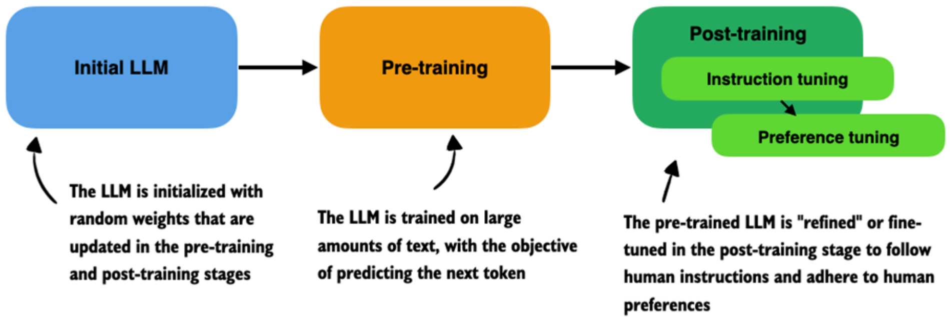

Initial LLM
Pre-training
Post-training
Instruction tuning
Preference tuning
The LLM is initialized with random weights that are updated in the pre-training and post-training stages
The LLM is trained on large amounts of text, with the objective of predicting the next token
The pre-trained LLM is "refined" or fine-tuned in the post-training stage to follow human instructions and adhere to human preferences

Figure 1.2 Overview of a typical LLM training pipeline. The process begins with an initial model initialized with random weights, followed by pre-training on large-scale text data to learn language patterns by predicting the next token. Post-training then refines the model through instruction fine-tuning and preference fine-tuning, which enables the LLM to follow human instructions better and align with human preferences.In the pre-training stage, LLMs are trained on massive amounts (many terabytes) of unlabeled text, which includes books, websites, research articles, and many other sources. The pre-training objective for the LLM is to learn to predict the next word (or token) in these texts.

In the pre-training stage of a typical LLM training pipeline, as shown in figure 1.2, LLMs are trained on massive amounts (many terabytes) of unlabeled text, which includes books, websites, research articles, and many other sources. The pre-training objective (goal) for the LLM is to learn to predict the next word (i.e., *token*) in these texts.

WORDS AND TOKENS

A token is a small unit of text that a language model processes. A token can be a full word, part of a word, or even punctuation, depending on how the text is split by a so-called tokenizer.

For example, the sentence "An LLM can be useful." might be broken into tokens like "An", " L", "LM", " can", " be", " useful", and "." by a common tokenizer. These tokens are then converted into numerical IDs that the model can ingest.

A tokenizer is a model or tool that is not directly part of the LLM itself but is nonetheless a critical component of the LLM text processing and generation pipeline. We will see how tokenization works in practice in the next chapter.

LLMs become highly capable when pre-trained on massive datasets, which typically involves several terabytes of text (equivalent to around 300 to 400 billion tokens). This training requires thousands of GPUs running for many months and can cost millions of dollars. Here, "capable" means that the LLMs begin to generate text that closely resembles human writing. Also, to some extent, pre-trained LLMs will begin to exhibit so-called emergent properties, which means that they will be able to perform tasks that they were not explicitly trained to do, including translation, code generation, and so on.

However, these pre-trained models merely serve as base models for the post-training stage, which uses two key techniques: *supervised fine-tuning* (often abbreviated as *SFT* in the literature and also known as *instruction tuning*) and *preference tuning* to teach LLMs to respond to user queries, which are illustrated in figure 1.3.

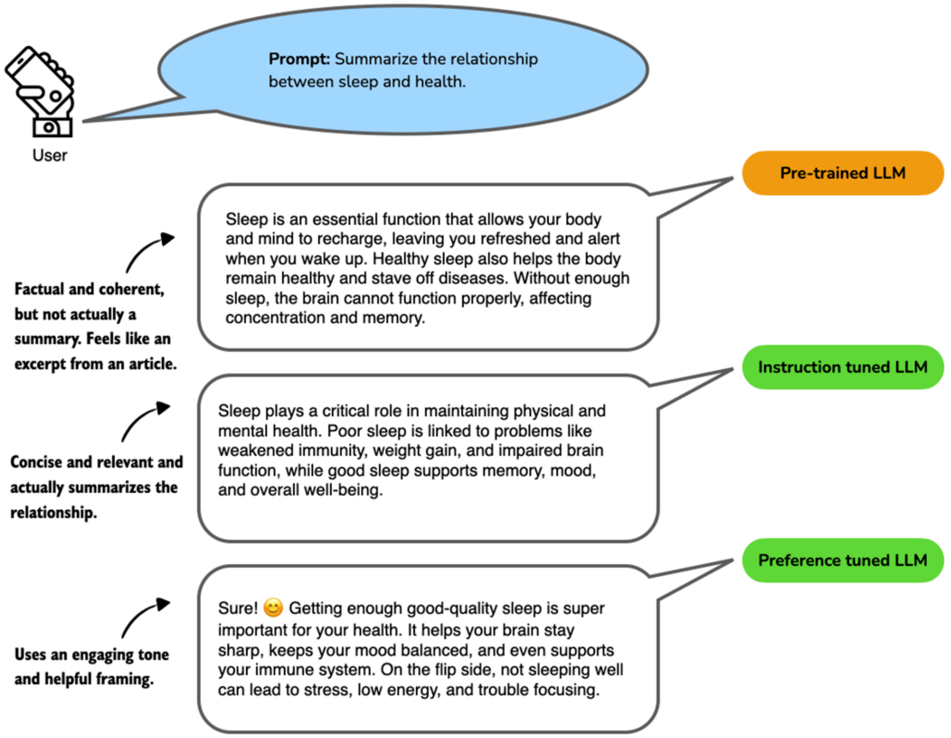

User
Prompt: Summarize the relationship between sleep and health.
Pre-trained LLM
Sleep is an essential function that allows your body and mind to recharge, leaving you refreshed and alert when you wake up. Healthy sleep also helps the body remain healthy and stave off diseases. Without enough sleep, the brain cannot function properly, affecting concentration and memory.
Factual and coherent, but not actually a summary. Feels like an excerpt from an article.
Instruction tuned LLM
Sleep plays a critical role in maintaining physical and mental health. Poor sleep is linked to problems like weakened immunity, weight gain, and impaired brain function, while good sleep supports memory, mood, and overall well-being.
Concise and relevant and actually summarizes the relationship.
Preference tuned LLM
Sure! 😊 Getting enough good-quality sleep is super important for your health. It helps your brain stay sharp, keeps your mood balanced, and even supports your immune system. On the flip side, not sleeping well can lead to stress, low energy, and trouble focusing.
Uses an engaging tone and helpful framing.

Figure 1.3 Example responses from a language model at different training stages. The prompt asks for a summary of the relationship between sleep and health. The pre-trained LLM produces a relevant but unfocused answer without directly following the instructions. The instruction-tuned LLM generates a concise and accurate summary aligned with the prompt. The preference-tuned LLM further improves the response by using a friendly tone and engaging language, which makes the answer more relatable and user-centered.

As shown in figure 1.3, instruction tuning improves an LLM's capabilities of personal assistance-like tasks like question-answering, summarizing and translating text, and many more. The preferences tuning stage then refines these capabilities. As the term implies, preference tuning helps tailor responses to user preferences. (Some readers may be familiar with terms like *Reinforcement Learning Human Feedback or RLHF*, which are specific techniques to implement preference tuning.)

In short, we can think of pre-training as "raw language prediction" (via next-token prediction) that gives the LLM some basic properties and capabilities to produce coherent texts. The post-training stage then improves the task understanding of LLMs via instruction tuning and refines the LLM to create answers with preferred stylistic choices via preference tuning.

These pre-training and post-training stages mentioned above are covered in my book "Build A Large Language Model (From Scratch)" (http://mng.bz/orYv) published by Manning. The book you are reading now does not require detailed knowledge of these stages. Concretely, in the next chapter, we will load a model that has already undergone the expensive pre-training and post-training stages mentioned above, so that we can focus on the methodology that is specific to reasoning models in the subsequent chapters.

1.3 Modeling language through pattern matching

As mentioned in the previous section, during pre-training, LLMs are exposed to vast quantities of text and learn to predict the next token by identifying and reproducing statistical associations in that data. This process enables them to generate fluent and coherent text, but it is fundamentally rooted in surface-level correlations rather than deep understanding.

LLMs respond to prompts by generating continuations that are statistically consistent with the patterns seen during training. In essence, they match patterns between input and output, rather than deducing answers through logical inference.

Consider the following example:

Prompt: "The capital of Germany is…"

Completion: "Berlin"

An LLM producing the answer "Berlin" is (likely) not logically deducing the answer. Instead, it is recalling a strong statistical association learned from training data. This behavior reflects what we refer to as pattern matching, which means that the model completes text based on learned correlations and not by applying structured reasoning steps.

But what about tasks that go beyond pattern recognition, i.e., tasks where a correct answer depends on drawing conclusions from given facts? This brings us to a different kind of capability: logical reasoning.

Logical reasoning involves systematically deriving conclusions from premises using rules or structured inference. Unlike pattern matching, it depends on intermediate reasoning steps and the ability to recognize contradictions or draw implications based on formal relationships.

Consider the following prompt as an example:

```
Prompt: "All birds can fly. A penguin is a bird. Can a penguin fly?"
```

A person, or a system capable of reasoning, would notice that something in the prompt example above is off. From the first two facts, you might conclude that a penguin should be able to fly. But if you also know that penguins cannot fly, there is a contradiction, as depicted in figure 1.4. A reasoning system would catch this conflict and recognize that either the first statement is too broad or that penguins are an exception.

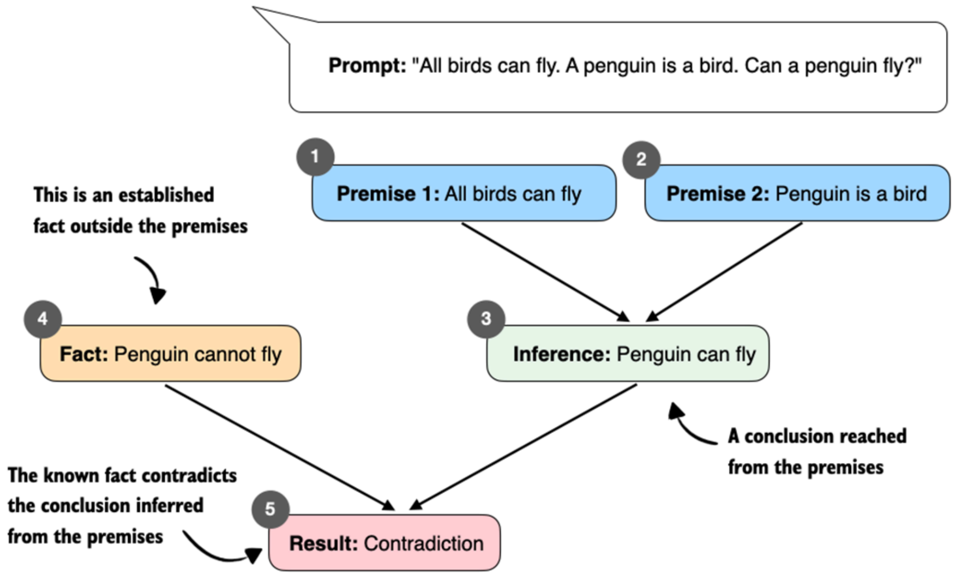

Prompt: "All birds can fly. A penguin is a bird. Can a penguin fly?"
1 Premise 1: All birds can fly
2 Premise 2: Penguin is a bird
This is an established fact outside the premises
4 Fact: Penguin cannot fly
3 Inference: Penguin can fly
A conclusion reached from the premises
The known fact contradicts the conclusion inferred from the premises
5 Result: Contradiction

Figure 1.4 Illustration of how contradictory premises lead to a logical inconsistency. From "All birds can fly" and "A penguin is a bird," we infer "Penguin can fly." This conclusion conflicts with the established fact "Penguin cannot fly," which results in a contradiction.

Figure 1.4 shows how a system based on logical reasoning could process the previously introduced "All birds can fly..." prompt.

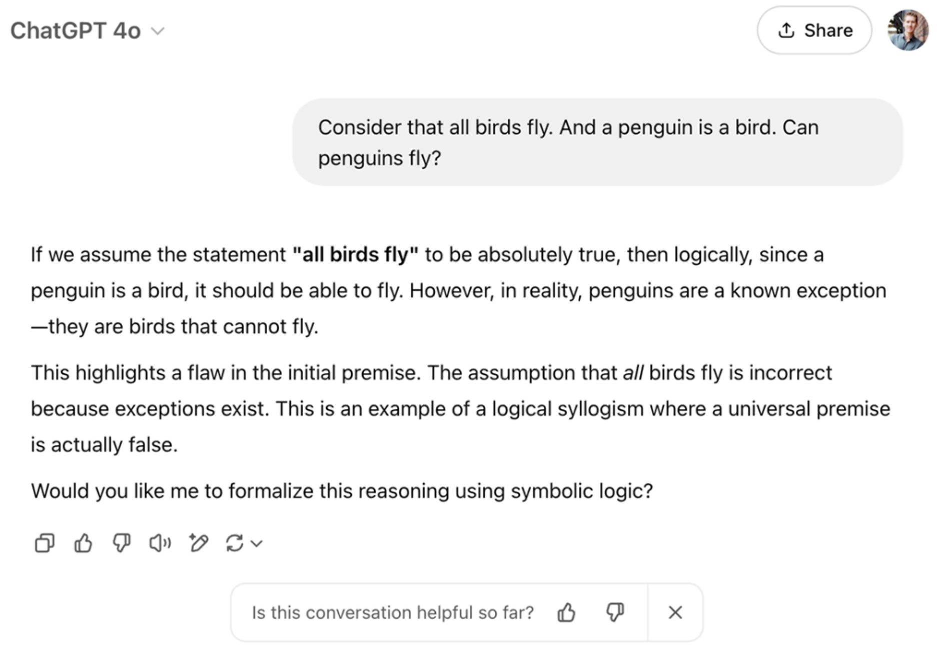
In contrast, a statistical (pattern-matching) LLM does *not* explicitly track contradictions as shown in figure 1.4 but instead predicts based on learned text distributions. For instance, if information such as "All birds can fly" is reinforced strongly in training data, the model may confidently answer: "Yes, penguins can fly."

In the next section, we will look at a concrete example of how an LLM handles this "All birds can fly..." prompt.

1.4 Simulating reasoning without explicit rules

In the previous section, we saw how contradictory premises can lead to logical inconsistencies. A conventional LLM does not explicitly track contradictions but generates responses based on learned text distributions.

Let's see a concrete example, shown in figure 1.5, of how a non-reasoning-enhanced LLM like OpenAI's ChatGPT 4o responds to the "All birds can fly..." prompt discussed in the previous section.



Consider that all birds fly. And a penguin is a bird. Can penguins fly?

If we assume the statement **"all birds fly"** to be absolutely true, then logically, since a penguin is a bird, it should be able to fly. However, in reality, penguins are a known exception —they are birds that cannot fly.

This highlights a flaw in the initial premise. The assumption that *all* birds fly is incorrect because exceptions exist. This is an example of a logical syllogism where a universal premise is actually false.

Would you like me to formalize this reasoning using symbolic logic?

Figure 1.5 An illustrative example of how a language model (ChatGPT 4o) appears to "reason" about a contradictory premise.

The example in figure 1.5 shows that ChatGPT 4o appears to answer correctly even though ChatGPT 4o is not considered a reasoning model, unlike OpenAI's other offerings like ChatGPT o1, o3, o4, and more recent GPT-5, which have been explicitly developed with reasoning methodology.

So, how did the 4o model generate its answer? Does this mean ChatGPT 4o explicitly reasons logically? No, not necessarily. However, at least 4o is highly effective at simulating logical reasoning in familiar contexts.

ChatGPT 4o does not implement explicit contradiction-checking and instead generates answers based on probability-weighted patterns. This approach works well enough if training data includes many instances correcting the contradiction (e.g., text like "penguins cannot fly") so that the model learns a statistical association between "penguins" and "not flying." As we see in figure 1.5, this allows the model to answer correctly without explicitly implementing rule-based or explicit logical reasoning methodologies.

In other words, the model recognizes the contradiction implicitly because it has frequently encountered this exact reasoning scenario during training. This effectiveness relies heavily on statistical associations built from abundant exposure to reasoning-like patterns in training data.

So, even when a conventional LLM seems to perform logical deduction as shown in figure 1.5, it's not executing explicit, rule-based logic but is instead leveraging patterns from its vast training data.

Nonetheless, ChatGPT 4o's success here is a great illustration of how powerful implicit pattern matching can become when trained at a massive scale. However, these types of pattern-based reasoning models usually struggle in scenarios where:

- The logical scenario is novel (not previously encountered in training data).
- Reasoning complexity is high, involving intricate, multi-step logical relationships.
- Structured reasoning is required, and no direct prior exposure to similar reasoning patterns exists in training data.

LOGICAL REASONING AND CURRENT REASONING LLM OFFERINGS

While ChatGPT 4o is not officially labeled as a reasoning model, OpenAI offers several dedicated reasoning models, including o1, o3, o4, and GPT-5. Moreover, other companies have been developing LLMs with explicit reasoning capabilities. As of this writing, popular examples include Anthropic's Claude 4, xAI's Grok 3, Google's Gemini 2.5, DeepSeek's R1, Alibaba's Qwen3, and many more. The techniques employed by these models are the focus of this book. As we will see, this is achieved without implementing a rule-based reasoning pipeline, as shown in figure 1.4. Instead, the LLM learns or improves its reasoning capabilities as a result of the modified inference and training methodologies.

LOGICAL REASONING AND RULE-BASED SYSTEMS

Why are explicit rule-based systems not more popular? Rule-based systems were used widely in the '80s and '90s for medical diagnosis, legal decisions, and engineering. They are still used in critical domains (medicine, law, aerospace), which often require explicit inference and transparent decision processes. However, they are hard to implement as they largely rely on human-crafted heuristics. However, deep neural networks (like LLMs) prove to be great and flexible at many tasks when trained at scale.

We might say that LLMs simulate logical reasoning through learned patterns, and we can improve it further with specific reasoning methods that include inference-compute scaling and post-training strategies, but they are not explicitly executing any rule-based logic internally.

Moreover, it's worth mentioning that reasoning in LLMs exists on a spectrum. This means that even before the advent of dedicated reasoning models such as ChatGPT o1 and DeepSeek-R1, LLMs were capable of simulating reasoning behavior. For instance, these models exhibited behaviors aligning with our earlier definition, such as generating intermediate steps to arrive at correct conclusions. What we now explicitly label a "reasoning model" is essentially a more refined version of this capability. And these improved reasoning capabilities are achieved by leveraging specific inference-compute scaling techniques and targeted post-training methods designed to improve and reinforce reasoning behavior.

The rest of this book focuses specifically on these advanced methods that improve LLMs to solve complex problems, helping you better understand how to improve the implicit reasoning capabilities in LLMs.

1.5 Improving reasoning with training and inference techniques

Reasoning in the context of LLMs became popular in the public eye with the announcement of OpenAI's ChatGPT o1 on September 12, 2024, which popularized the concept of reasoning in LLMs. In the announcement article (https://openai.com/index/introducing-openai-o1-preview/), OpenAI mentioned that "We've developed a new series of AI models designed to spend more time thinking before they respond."

Furthermore, OpenAI wrote: "These enhanced reasoning capabilities may be particularly useful if you're tackling complex problems in science, coding, math, and similar fields."

While the training and implementation details of ChatGPT o1 are not publicly available, the common perception is that the o1 model is based on one of the predecessors, like GPT-4, but uses extensive inference-compute scaling (more on that later in this chapter) to achieve these enhanced reasoning capabilities.

A few months later, in January 2025, DeepSeek released the DeepSeek-R1 model and technical report (https://arxiv.org/abs/2501.12948), which details training methodologies to develop reasoning models, which made big waves as they not only made freely and openly available a model that competes with and exceeds the performance of the proprietary o1 model but also shared a blueprint on how to train such model.

This book aims to explain how these methodologies used to develop reasoning models work by implementing similar methods from scratch.

The different approaches to developing and improving an LLM's reasoning capabilities can be grouped into three broad categories, as illustrated in figure 1.6.

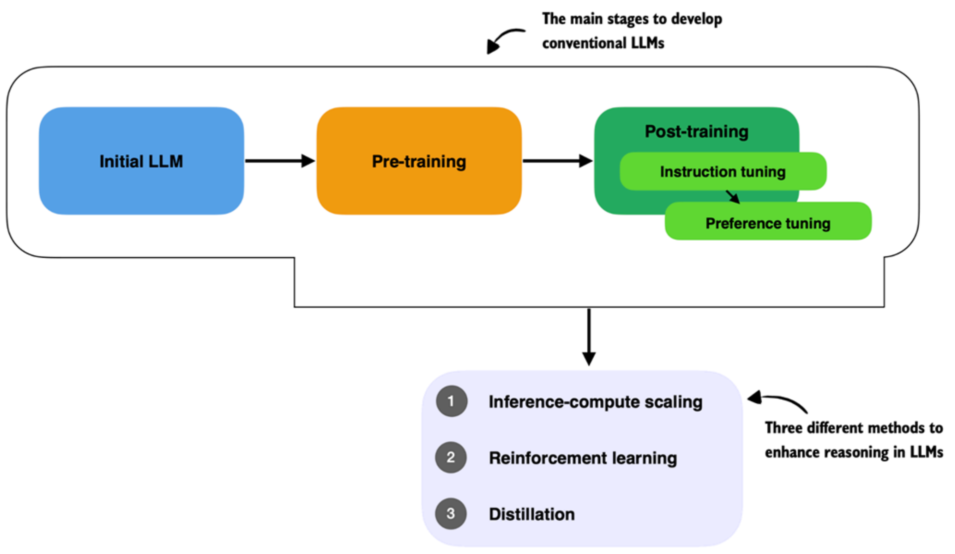

The main stages to develop conventional LLMs
Initial LLM
Pre-training
Post-training
Instruction tuning
Preference tuning
1 Inference-compute scaling
2 Reinforcement learning
3 Distillation
Three different methods to enhance reasoning in LLMs

Figure 1.6 Three approaches commonly used to improve reasoning capabilities in LLM). These methods (inference-compute scaling, reinforcement learning, and distillation) are typically applied after the conventional training stages (initial model training, pre-training, and post-training with instruction and preference tuning).

As illustrated in figure 1.6, these methods are applied to LLMs that have undergone the conventional pre-training and post-training phases, including instruction and preference tuning.

1. **Inference-time compute scaling.** *Inference-time compute scaling* (also often called *inference compute scaling*, *test-time scaling*, or other variations) includes methods that improve model reasoning capabilities at inference time (when a user prompts the model) without training or modifying the underlying model weights. The core idea is to trade off increased computational resources for improved performance, which helps make even fixed models more capable through techniques such as chain-of-thought reasoning, and various sampling procedures. This topic will be the focus of chapter 4.

2. **Reinforcement learning.** *Reinforcement learning* (*RL*) refers to training methods that improve a model's reasoning capabilities by encouraging it to take actions that lead to high reward signals. These rewards can be broad, such as task success or heuristic scores, or they can be narrowly defined and verifiable, such as correct answers in math problems or coding tasks.
   Unlike scaling compute at inference time, which can improve reasoning performance without modifying the model, RL updates the model's weights during training. This enables the model to learn and refine reasoning strategies through trial and error, based on the feedback it receives from the environment. We will explore RL in more detail in chapter 5.

REINFORCEMENT LEARNING FOR REASONING AND PREFERENCE TUNING

In the context of developing reasoning models, it is important to distinguish the RL approach here from reinforcement learning with human feedback (RLHF), which is used during preference tuning when developing a conventional LLM as illustrated previously in figure 1.2. RLHF incorporates explicit human evaluations or rankings of model outputs as reward signals, directly guiding the model toward human-preferred behaviors. In contrast, RL in the context of reasoning models typically relies on automated or environment-based reward signals, which can be more objective but potentially less aligned with human preferences. For instance, RL in a reasoning model development pipeline might train a model to excel at mathematical proofs by providing explicit rewards for correctness. In contrast, RLHF would involve human evaluators ranking various responses to encourage outputs that align closely with human standards and subjective preferences.

3. **Supervised fine-tuning and model distillation.** *Distillation* involves transferring complex reasoning patterns learned by powerful, larger models into smaller or more efficient models. Within the context of LLMs, this typically means performing *supervised fine-tuning* (*SFT*) using high-quality labeled instruction datasets generated by a larger, more capable model. This technique is commonly referred to as *knowledge distillation* or simply *distillation* in LLM literature. However, it's important to note that this differs slightly from traditional knowledge distillation in deep learning, where a smaller ("student") model typically learns from both the outputs and the logits produced by a larger ("teacher") model. This topic is discussed further in Chapter 6.

SUPERVISED FINE-TUNING FOR REASONING AND CONVENTIONAL LLM DEVELOPMENT

The SFT technique here is similar to the SFT technique used when developing a conventional LLM, except that the training examples are derived from a model explicitly developed for reasoning. Consequently, the training examples here focus more on reasoning tasks and typically include intermediate reasoning steps

1.6 Building reasoning models from scratch

Following the release of DeepSeek-R1 in January 2025, improving the reasoning abilities of LLMs has become one of the hottest topics in AI, and for good reason. Stronger reasoning skills allow LLMs to tackle more complex problems, making them more capable across various tasks users care about.

This shift is also reflected in a February 12, 2025, statement from OpenAI's CEO:

> "We will next ship GPT-4.5, the model we called Orion internally, as our last non-chain-of-thought model. After that, a top goal for us is to unify o-series models and GPT-series models by creating systems that can use all our tools, know when to think for a long time or not, and generally be useful for a very wide range of tasks."

The quote above underlines the major shift from leading LLM providers towards reasoning models, where "chain-of-thought" refers to a prompting technique that guides language models to reason step-by-step to improve their reasoning capabilities, which we will cover in more detail in Chapter 4.

Also noteworthy is the mention of knowing "when to think for a long time or not." This hints at an important design consideration: reasoning is not always necessary or desirable

For instance, reasoning models are designed to be good at complex tasks such as solving puzzles, advanced math problems, and challenging coding tasks. However, they are not necessary for simpler tasks like summarization, translation, or knowledge-based question answering. In fact, using reasoning models for everything can be inefficient and expensive. For instance, reasoning models are typically more expensive to use, more verbose, and sometimes more prone to errors due to "overthinking." Also, here, the simple rule applies: Use the right tool (or type of LLM) for the task.

Why are reasoning models more expensive than non-reasoning models? Primarily because they tend to produce longer outputs, due to the intermediate reasoning steps that explain how an answer is derived. As illustrated in Figure 1.7, LLMs generate text one token at a time. Each new token requires a full forward pass through the model. So, if a reasoning model produces an answer that is twice as long as that of a non-reasoning model, it will require twice as many generation steps, resulting in twice the computational cost. This also directly impacts cost in API usage, where billing is often based on the number of tokens processed and generated.

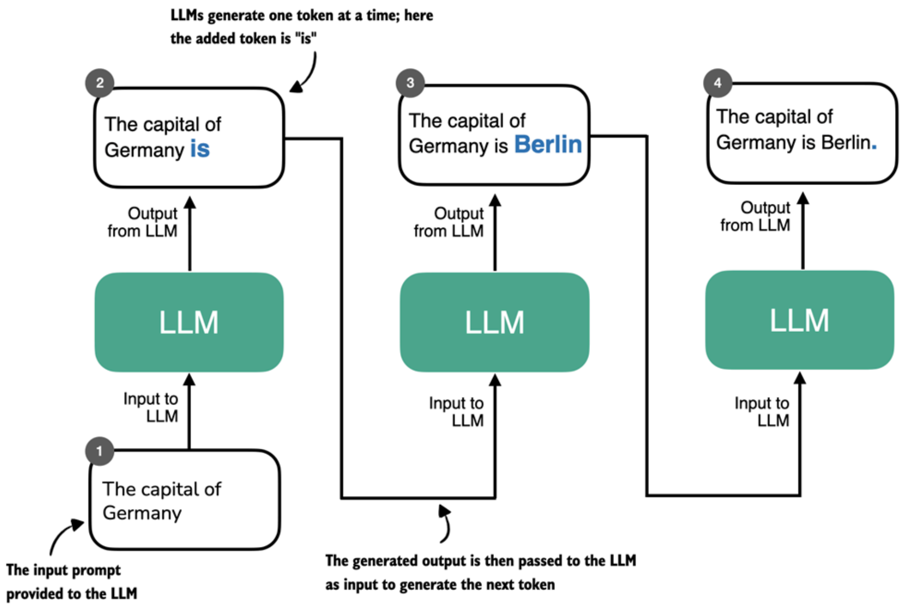

LLMs generate one token at a time; here the added token is "is"
2
The capital of Germany is
Output from LLM
LLM
Input to LLM
1
The capital of Germany
The input prompt provided to the LLM
3
The capital of Germany is Berlin
Output from LLM
LLM
Input to LLM
The generated output is then passed to the LLM as input to generate the next token
4
The capital of Germany is Berlin.
Output from LLM
LLM
Input to LLM

Figure 1.7 Token-by-token generation in an LLM. At each step, the LLM takes the full sequence generated so far and predicts the next token, which may represent a word, subword, or punctuation mark depending on the tokenizer. The newly generated token is appended to the sequence and used as input for the next step. This iterative decoding process is used in both standard language models and reasoning-focused models.

This directly highlights the importance of implementing LLMs and reasoning methods from scratch. It's one of the best ways to understand how they work. And if we understand how LLMs and these reasoning models work, we can better understand these trade-offs.

1.7 A roadmap to reasoning models from scratch

Now that we have discussed reasoning in LLMs from a bird's-eye view, the subsequent chapters will guide you through the process of coding and applying reasoning methods from scratch. We will tackle this in multiple stages, as outlined in figure 1.8.

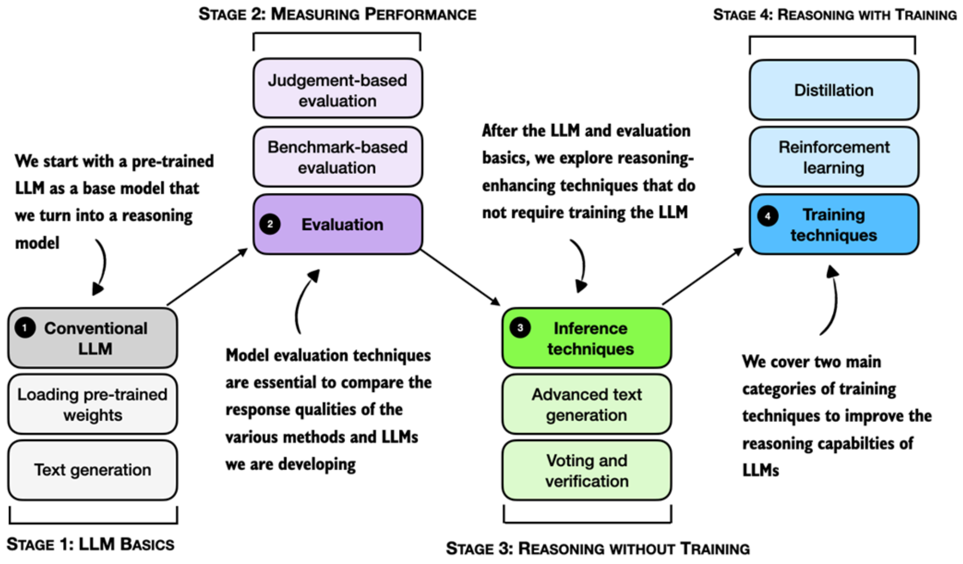

Stage 2: Measuring Performance
Judgement-based evaluation
Benchmark-based evaluation
2 Evaluation
Stage 4: Reasoning with Training
Distillation
Reinforcement learning
4 Training techniques
We start with a pre-trained LLM as a base model that we turn into a reasoning model
After the LLM and evaluation basics, we explore reasoning-enhancing techniques that do not require training the LLM
1 Conventional LLM
Loading pre-trained weights
Text generation
Stage 1: LLM Basics
Model evaluation techniques are essential to compare the response qualities of the various methods and LLMs we are developing
3 Inference techniques
Advanced text generation
Voting and verification
Stage 3: Reasoning without Training
We cover two main categories of training techniques to improve the reasoning capabilties of LLMs

Figure 1.8 A mental model of the main reasoning model development stages covered in this book. We start with a conventional LLM as base model (stage 1). In stage 2, we cover evaluation strategies to track the reasoning improvements introduced via the reasoning methods in stages 3 and 4.

As shown in figure 1.8, we cover the reasoning model development in several stages. In stage 1 (next chapter), we load a conventional LLM that has already undergone the basic pre-training and instruction fine-tuning stages. Then, in stage 2, we cover common methods for evaluating LLMs and reasoning capabilities, so that we can measure the improvements we make when we apply reasoning-enhancing methods in stages 3 and 4.

Stage 3 covers inference techniques that can improve the response quality and reasoning behavior of LLMs. Note that these techniques can be applied to improve any LLM, conventional LLMs and LLMs that have been trained as reasoning models. Stage 4 will introduce training methods to develop reasoning models.

I hope you are as enthusiastic as I am about the journey ahead!

1.8 Summary

- Conventional LLM training occurs in several stages:
  - Pre-training, where the model learns language patterns from vast amounts of text.
  - Instruction fine-tuning, which improves the model's responses to user prompts.
  - Preference tuning, which aligns model outputs with human preferences.

- Reasoning methods are applied on top of a conventional LLM.
- Reasoning in LLMs involves systematically solving multi-step tasks using intermediate steps (chain-of-thought).

- Reasoning in LLMs is different from rule-based reasoning and it also likely works differently than human reasoning; it's currently believed that reasoning in LLMs relies on statistical pattern matching.
- Pattern matching in LLMs relies purely on statistical associations learned from data, which enables fluent text generation but lacks explicit logical inference.
- Improving reasoning in LLMs can be achieved through:
  - Inference-time compute scaling, enhancing reasoning without retraining (e.g., chain-of-thought prompting).
  - Reinforcement learning, training models explicitly with reward signals.
  - Supervised fine-tuning and distillation, using examples from stronger reasoning models.
- Building reasoning models from scratch provides practical insights into LLM capabilities, limitations, and computational trade-offs.

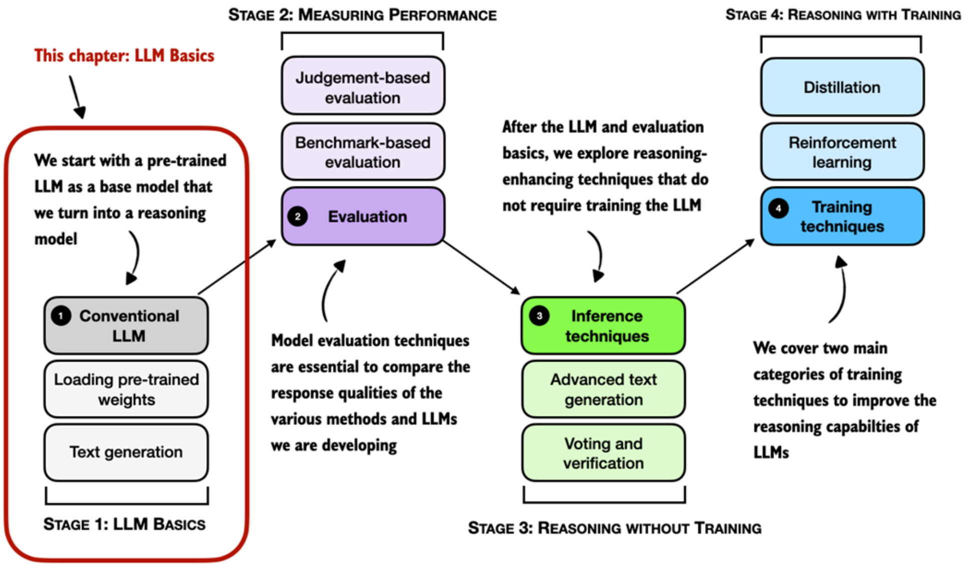
2 Generating text with a pre-trained LLM

This chapter covers

- **Setting up the code environment for working with LLMs**
- **How to use a tokenizer to prepare input text for an LLM**
- **The step-by-step process of text generation using a pre-trained LLM**
- **Caching and compilation techniques for speeding up LLM text generation**

In the previous chapter, we discussed the difference between *conventional large language models* (*LLMs*) and *reasoning models*. Also, we introduced several techniques to improve the reasoning capabilities of LLMs. These reasoning techniques are usually applied on top of a conventional (base) LLM.

In this chapter, we will lay the groundwork for the upcoming chapters and load such a conventional LLM on top of which we can apply reasoning techniques in subsequent chapters, as illustrated in figure 1. This conventional LLM is an LLM that has already been pre-trained to generate general texts (but it has not been specifically trained or enhanced for reasoning).



Stage 2: Measuring Performance
Stage 4: Reasoning with Training
This chapter: LLM Basics
We start with a pre-trained LLM as a base model that we turn into a reasoning model
1 Conventional LLM
Loading pre-trained weights
Text generation
Stage 1: LLM Basics
Judgement-based evaluation
Benchmark-based evaluation
2 Evaluation
Model evaluation techniques are essential to compare the response qualities of the various methods and LLMs we are developing
After the LLM and evaluation basics, we explore reasoning-enhancing techniques that do not require training the LLM
3 Inference techniques
Advanced text generation
Voting and verification
Stage 3: Reasoning without Training
Distillation
Reinforcement learning
4 Training techniques
We cover two main categories of training techniques to improve the reasoning capabilties of LLMs

Figure 2.1 A mental model depicting the four main stages of developing a reasoning model. This chapter focuses on stage 1, loading a conventional LLM and implementing the text generation functionality.

In addition to setting up the coding environment and loading a pre-trained LLM, you will learn how to use a *tokenizer* to prepare text input for the model. As illustrated in figure 2.1, you will also implement a text generation function, enabling practical use of the LLM to generate text. This functionality will be used and further improved in later chapters.

2.1 Introduction to LLMs for text generation

In this chapter, we implement all the necessary LLM essentials, from setting up our coding environment and loading a pre-trained LLM to generating text that we will reuse and build upon in this book. In this sense, this chapter can be understood as a setup chapter.

This LLM will be capable of following basic instructions and generating coherent text, as illustrated in figure 2.2.