CHAPTER 11
Looking Ahead

Human history only makes sense on a logarithmic scale. It took humans countless eons to figure out farming, millennia beyond that to invent writing, centuries more to invent the steam engine, and decades more to invent the automobile, computer, and smartphone. Just a few years after that, around 2012, deep learning appeared on the scene.

OpenAI's GPT-2 was announced in 2019, and then ChatGPT was announced in 2022. This ignited an explosion of development around LLMs. Many companies have jumped into the fray—Anthropic, Google, Microsoft, Meta, xAI, NVIDIA, Mistral, and more—all building new LLMs that have leap-frogged the previous ones in capability, capacity, and speed. In mere months, LLMs have morphed from document completion engines, to chat engines, to agents that can interact with the outside world.

Buckle up, readers. If you think the pace of change is fast now, then just wait, it's only going to get faster. (Maybe that Ray Kurzweil guy was on to something!) In this final chapter, let's look ahead to some of the developments on our horizon and how they will change your work as a prompt engineer.

Multimodality

There is a huge push toward the use of multimodal models. OpenAI kicked off this trend with GPT-4, which was able to process images as part of the prompt. Although OpenAI has not disclosed details about how exactly the model works, most likely, it closely follows the methods published in academic literature.

In one such method, a convolutional network is used to convert image features into embedding vectors of the same dimensions as those used for text tokens. The image vectors are imbued with positional information so that the relationships among the features in the image are retained. The image and text vectors are then concatenated. Finally, the transformer architecture processes this information in much the same way that text-only LLMs process pure text (see Chapter 2). Multimodality can naively be extended to video input—all you have to do is sample images from the video, as demonstrated in this OpenAI cookbook.

As they mature, multimodal models are going to be extremely useful in domains that can't be captured by text. For example, it's easy to imagine how these models can be used to make the world much more accessible to a person with vision impairment. A vision model could help them read signs, find buildings, and navigate unfamiliar environments.

Another reason that multimodal models are important is because they give the models access to a large volume of rich training data. Over the past few years, there has been increasing concern that we might actually run out of training data! The models are large enough that they can learn increasingly intricate details about the world. However, if we overtrain with a set of data that is too small, then these models can overfit—effectively memorizing text rather than modeling how the world works. Amazingly, literally *the text of the entire public internet* may not be enough for the next generation of large models.

However, when we incorporate images and video into the training, we gain access to vastly more content. Moreover, the image and video content carry a very different type of information that can help models better understand the world around them; with access to images, it should become much simpler for models to understand tasks related to spatial reasoning, social cues, physical common sense, and much more.

As a prompt engineer, when you build future LLM applications, you are likely to include images and videos in the prompt—and even though they constitute a completely different form of information, you can make use of some of the lessons in this book when dealing with them. Remember to include only images that are relevant to the conversation at hand so that the model doesn't get distracted. Frame the images with text that properly introduces their role in the conversation and make use of patterns and motifs that were in the training data. For instance, don't introduce a new type of diagram to convey information when there is a common format that is more readily available on the internet.

User Experience and User Interface

The UI of many consumer applications is currently moving toward conversational interactions. It makes sense, right? Humans have been speaking to each other for 200,000 years but only clicking buttons on screens for the past 40. In this section, we'll focus on a new element of the conversation that has caught our attention—artifacts—or, as we like to call them, *stateful objects of discourse*.

Think about it. In day-to-day collaboration with other humans, we often talk about a *thing*—the *object of discourse*. And as we talk about it, we can talk about how we want to change it, we can actually modify it, and we can talk about how it has changed over time—meaning we can talk about its state. Pair programming is a great example here. The files are the objects of discourse, and during pairing, we can change them and talk about how they are changing.

In most chat applications today, the assistants don't address the object of the conversation in a stateful way. If you ask ChatGPT to write a function and then later modify it, it can't go back and update the contents of the function. Instead, it rewrites the function over and over again, from scratch. Rather than having one object whose state has evolved, ChatGPT writes *N* objects into the conversation.

What's more, it is difficult to specify which object you're talking about, especially if you have multiple objects in play. Which function were you talking about? Which version? These problems make it difficult to work with the assistant *on* something rather than just having a conversation in which ideas are expected to fly by and go out of scope.

As we were wrapping up this book, Anthropic introduced *Artifacts*, which represents a step in the direction of stateful objects of discourse. In a conversation with Anthropic's Claude, the Artifact *is* the stateful object of discourse. It can be an SVG image, an HTML file, a mermaid diagram, code, or any other type of text fragment. During a conversation, the user works with the assistant to modify the Artifact until it reaches the user's expectations. And while the back-and-forth conversation is captured in a transcript on the left side of the screen, the Artifact they are discussing remains—statefully—on the right side of the screen (see Figure 11-1). If the user asks for the Artifact to be modified, then the state of the Artifact is updated in place rather than regurgitated over and over again in the transcript.

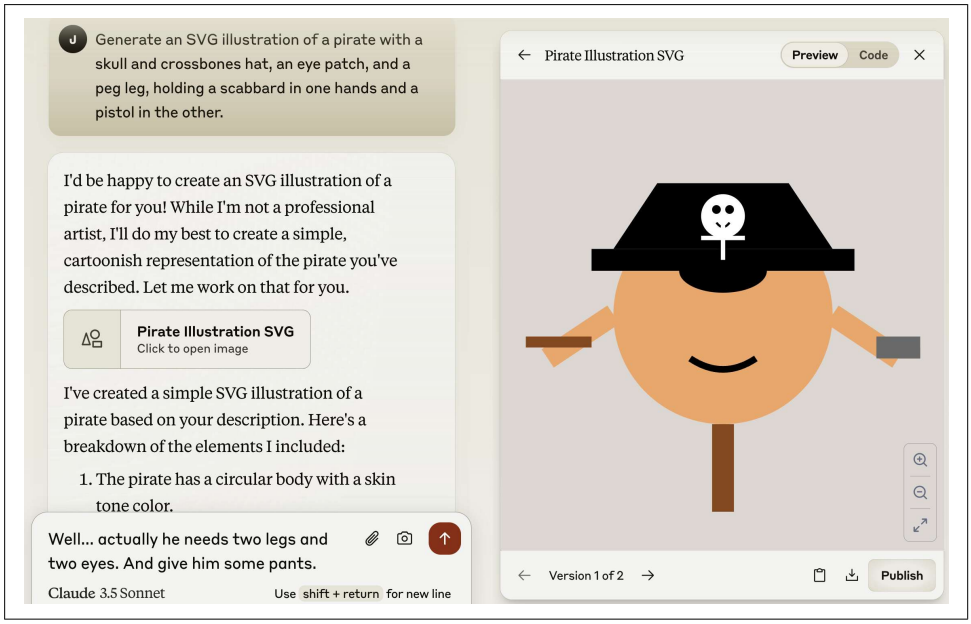

Figure 11-1. Working with Claude to draw a peg-legged pirate with a patch over his eye while statefully discussing the fact that the image appears to be missing actual legs and eyes

Claude's Artifacts paradigm is very close to what we have in mind, but there's still room for improvement. For instance, much of the change is just in the UI, rather than in the prompt engineering. When you ask for a change, Claude is still rewriting the entire Artifact from scratch; it just knows to put the Artifact in the right pane. This form of editing might not scale well to longer documents.

Additionally, it's not easy to interact with multiple Artifacts at once. Claude's interface assumes one Artifact at a time. If you start talking about a different Artifact, then the UI treats it as if it were just a different version of the previous interface. Another problem with multiple Artifacts is that it's hard to refer to them. It would be nice if both the UI and the prompt included shorthand names for the items.

Finally, Claude's interface and the prompt engineering (for that matter) don't allow the user to edit the Artifact. If you see a small problem that you could easily fix, the only way to address it is to tell the assistant to fix the problem for you (by retyping the whole file). It would be a better experience if the user could update the Artifact and then have this update reflected in the next prompt so that the model is aware of the change.

When building LLM interfaces in your own LLM applications, it can be a good idea to lean into a conversational interface since conversation is so intuitive for humans. But if so, you need to invest time to really get it right—it's easy to whip up something bare-bones with LLMs, but it will be a gimmicky distraction rather than something that provides true benefits, unless it's properly thought through and fleshed out.

Model designers are aware of this need, and they innovate to support it. Tools were a great improvement—they gave the assistants the ability to take action in the world. Artifacts are similarly useful—they allow conversations to be about *things* (i.e., stateful objects of discourse). What's next?

The conversational UI is also a great way to keep users in the loop. As we discussed in Chapter 8, models tend to stray off course if left to their own devices. But in a close conversational interaction, users can identify problems early and put the assistant back on course.

Intelligence

Has anyone noticed that LLMs are getting a lot smarter? Yeah…and they're going to keep getting smarter too. Let's look at some upcoming developments.

For one thing, we're getting smarter with our benchmarks. Benchmarks are problem sets with known answers that allow us to measure how well models perform relative to humans and relative to one another. At this point, several of the most useful benchmarks have saturated (see Figure 11-2), meaning that the leading models tend to ace them. This makes the benchmarks useless for evaluating model improvements. There are two reasons that models saturate the benchmarks: (1) models really are getting smarter—*a good thing*, and (2) models are cheating by training on the benchmarks—*a very bad thing*. The "cheating" isn't intentional; it's just that after a couple of years, information from benchmarks gets duplicated (verbatim or through descriptions) all over the internet and accidentally pulled into training.

To fix this, we in the AI community are being diligent in upgrading our benchmarks (for instance, on the Open LLM Leaderboard 2). We have also started using nonmemorizable benchmarks such as ARC-AGI, which is effectively a set of psychometric intelligence tests composed of patterns of shapes. They test how well the individual—or LLM—can understand and reproduce novel patterns, and it's impossible to memorize all of the test questions because they belong to a very large space of possible tests, they are algorithmically generated, and you can always generate more of them.

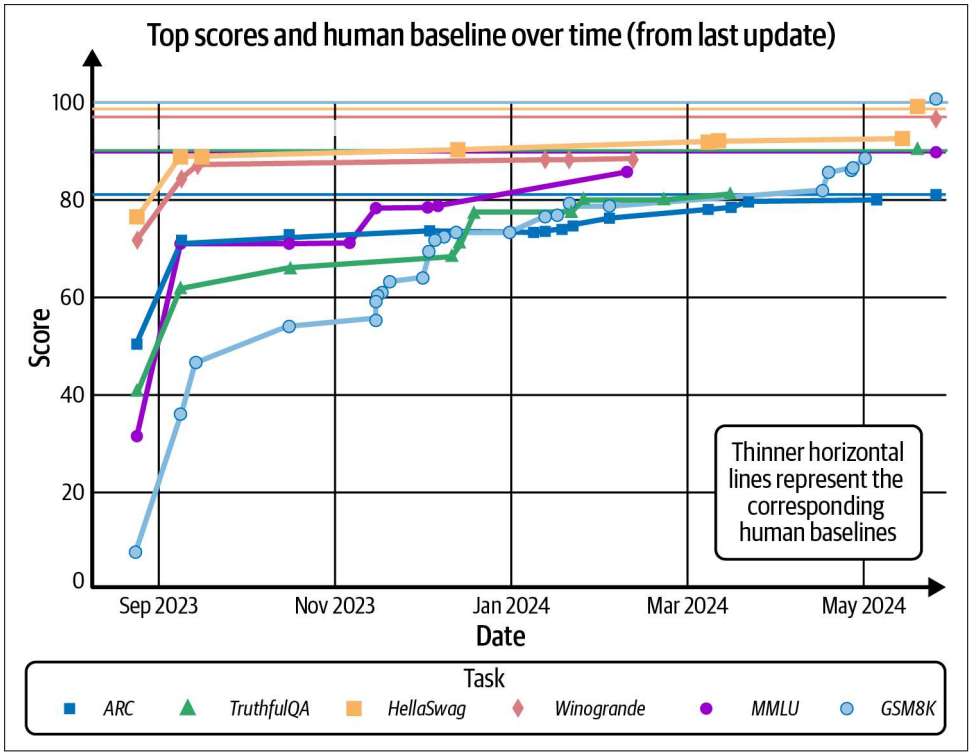

Figure 11-2. Popular benchmarks saturate over time, making them useless as benchmarks going forward

We're also getting smarter with model training. You can actually watch this happen when you use ChatGPT or its competitors because, thanks to better RLHF training (refer back to Chapter 3), the models are doing a much better job of expressing their chain-of-thought reasoning, which inevitably leads to more useful responses.

Next, we're getting more creative with training approaches. For example, large models generally don't utilize their full capacity, so, if you can figure out a way to convey knowledge to a small model, then you can effectively compress the information of the large model into the small model. A training approach known as *knowledge distillation* uses a large model as a "teacher" of a small model. Rather than training the small model to predict the next token, knowledge distillation trains the small model to mimic the large model by predicting the full set of probabilities for the next token. This richer set of training data allows smaller models to be trained quickly, with only a slight decrease in accuracy compared to their teacher models. In exchange for the hit in accuracy, these small models are significantly cheaper and faster than the large models they were trained from.

Besides training, model improvements will come from architectural innovation. Here, we name just a few. For one, models are getting smaller and faster through *quantization* approaches, in which, instead of representing parameters as 32-bit floating point numbers, you can approximate them with 8-bit parameters, reducing the model size considerably and correspondingly decreasing the cost and increasing the speed.

In your prompt-engineering work, you should be able to expect these trends to continue. If something is too expensive today, it will be cheaper tomorrow. If something is too slow today, it will be faster tomorrow. If something doesn't fit in the context today, it will fit tomorrow. And if the model isn't smart enough today, it will be tomorrow. However, always remember that even though models will get smarter, they will never be psychic. If the prompt doesn't contain the information that *you* would need to solve the problem, then it's probably insufficient for the model as well.

Conclusion

If we had to sum up the main lessons from this book, there would be two:

1. LLMs are nothing more than text completion engines that mimic the text they see during training.
2. You should empathize with the LLM and understand how it thinks.

Regarding the first lesson, when we started writing this book, the only models we had access to were completion models—give them a portion of a document (a.k.a. *the prompt*), and they would generate plausible text to complete the document. But then, the chat APIs came to dominate, tools came along next, and perhaps, Artifacts will be the next big thing. But even still, at their core, LLMs are just completing documents so that they resemble other documents the model has been taught to "like." It's just that now, the documents look like a chat transcript.

The prompt-engineering lesson here is to follow the well-trodden course (the Little Red Riding Hood principle from Chapter 4)—make your prompts follow the patterns and motifs seen in training data and you will be much more likely to get completions that are well behaved and easy to anticipate. For instance, you can format complex text as markdown, and if there is a standard document format for information you are communicating to the LLM, then you'll have better luck using that format than coming up with a new one the model has never seen.

Regarding the second lesson, on empathy, think of an LLM as your big, dumb mechanical friend who happens to know much of the content of the internet. Here are some things that will help you understand:

LLMs are easily distracted
: Don't fill up the prompt with useless information that—*cross your fingers*—might just help. Make sure every piece of information matters.

LLMs should be able to decipher the prompt
: If, as a human, you can't understand the fully rendered prompt, then there is a very high chance that the LLM will be equally confused.

LLMs need to be led
: Provide explicit instructions for what is to be accomplished and, when appropriate, provide examples demonstrating how the task should proceed.

LLMs aren't psychic
: As the prompt engineer, it's your job to make sure the prompt contains the information that the model needs to address the problem. Alternatively, give the model the tools and the instructions to retrieve it.

LLMs don't have internal monologues
: If the LLM is allowed to think about the problem out loud (in a chain of thought), then it will be much easier for it to come to a useful solution.

Hopefully, this book has given you all that you need to jump headlong into prompt engineering and LLM application development. Be assured, the accelerating change that we currently experience will continue. Since software will be easier to create, you will find more examples of highly individualized apps or even disposable apps. Applications will take on the nondeterministic nature of the LLMs, leading to more flexible and open-ended experiences. Development will change. You will work in tandem with an AI assistant to get your work done—if you don't already.

Whatever shape the world finds itself in, it will be a shape of your making. As a prompt engineer, you have the tools at hand and the know-how to build a future of your choosing. Embrace the acceleration. Keep experimenting. Stay flexible. In the words of the late Sir Terry Pratchett:

> The whole world is tap-dancing on quicksand. In this case, the prize goes to the best dancer.[1]

1 Terry Pratchett, *The Fifth Elephant* (New York: Doubleday, 1999)

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

Q

R

S

T

U

V

W

X

Y

Z

About the Authors

John Berryman is the founder and principal consultant of Arcturus Labs, where he specializes in LLM application development. His expertise helps businesses harness the power of advanced AI technologies. As an early engineer on GitHub Copilot, John contributed to the development of its completions and chat functionalities, working at the forefront of AI-assisted coding tools.

Before his work on Copilot, John built a varied career as a search engineer. His diverse experience includes helping to develop a next-generation search system for the US Patent Office, building search and recommendations for Eventbrite, and contributing to GitHub's code search infrastructure. John is also coauthor of *Relevant Search* (Manning), a book that distills his expertise in the field.

John's unique background, spanning both cutting-edge AI applications and foundational search technologies, positions him at the forefront of innovation in LLM applications and information retrieval.

Albert Ziegler has been designing AI-driven systems long before LLM applications became mainstream. As founding engineer for GitHub Copilot, he designed its prompt engineering system and helped inspire a wave of AI-powered tools and "Copilot" applications, shaping the future of developer assistance and LLM applications.

Today, Albert continues to push the boundaries of AI technology as Head of AI at XBOW, an AI cybersecurity company. There, he leads efforts blending large language models with cutting-edge security applications to secure the digital world of tomorrow.

Colophon

The animal on the cover of *Prompt Engineering for LLMs* is a banteng (*Bos javanicus*), a species of wild cattle in Southeast Asia. Banteng live in herds of one bull and many cows. They are brown or black with white stockings and light patches on their backsides. Both sexes have horns, but bulls are typically larger and darker. Adults banteng weigh up to 1,900 pounds and stand at 6 feet (hoof to shoulder).

The wild banteng is classified as Endangered on the IUCN Red List. Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Jose Marzan, based on an antique line engraving from *Meyers Kleines Lexicon*. The series design is by Edie Freedman, Ellie Volckhausen, and Karen Montgomery. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.