

O'REILLY®



Early  
Release

RAW &  
UNEDITED

# LLMOps

Managing Large Language Models  
in Production

Abi Aryan

# LLMOps

Managing Large Language Models in Production

**Abi Aryan**

**O'REILLY®**

[OceanofPDF.com](https://oceanofpdf.com)

# LLMOps

by Abi Aryan

Copyright © 2025 Abi Aryan. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North,  
Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Sarah Grey and Nicole Butterfield

Production Editor: Beth Kelly

Copyeditor: FILL IN COPYEDITOR

Proofreader: FILL IN PROOFREADER

Indexer: FILL IN INDEXER

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Kate Dullea

July 2025: First Edition

## Revision History for the Early Release

- 2025-01-29: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781098154202> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *LLMOps*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-098-15414-1

[FILL IN]

[OceanofPDF.com](http://OceanofPDF.com)

# Brief Table of Contents (*Not Yet Final*)

---

Chapter 1: Introduction to Large Language Models (available)

Chapter 2: Introduction to LLMOps (available)

*Chapter 3: Data Engineering for LLMs (unavailable)*

*Chapter 4: API-First LLM Deployment (unavailable)*

*Chapter 5: Model Domain Adaptation for LLM-Based Applications (unavailable)*

*Chapter 6: LLM-Based Applications (unavailable)*

*Chapter 7: Model Evaluation (unavailable)*

*Chapter 8: Governance: Monitoring, Privacy and Security (unavailable)*

*Chapter 9: Hardware Scaling: Infrastructure and Resource Management (unavailable)*

Chapter 10: The Future of LLMOps and Large Language Models (available)

[OceanofPDF.com](https://oceanofpdf.com)

# Chapter 1. Introduction to Large Language Models

---

## A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book. The GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at [sgrey@oreilly.com](mailto:sgrey@oreilly.com).

Large language models (LLMs) are the much-hyped rockstars of the AI world. They’re making headlines daily for their incredible abilities to generate text, hold conversations, and understand complex queries. However, as we said in the Preface, their rise to popularity is no accident; they’re transforming how we interact with technology and pushing the boundaries of what machine learning models can do.

But here’s the catch: while these models are impressive, scaling them up and managing them in production is no walk in the park. The leap from a research project to a fully fledged, reliable tool is filled with obstacles. We’re talking about meeting enormous computational requirements, managing complex data, and ensuring that everything runs smoothly and securely.

In Chapter 2 we’ll introduce LLMOps—a field dedicated to tackling these challenges. Think of LLMOps as the playbook for turning these models

from exciting prototypes into practical, high-performing systems. It's where theory meets practice, offering strategies, tools and frameworks to overcome the limitations and make LLMs work for us.

But before we dive into the nitty-gritty of LLM operations, it's important to understand why and how these models came to be. Knowing their origins and genealogy helps us appreciate the challenges we will face when predicting their behaviors in production.

The evolution of LLMs reflects a series of incremental innovations, each addressing specific limitations of previous models. Early models were limited in scope and required extensive human input for even basic tasks. With advancements in architecture, such as the shift from RNNs to transformers, and the scaling of model sizes, LLMs have become more sophisticated. This evolution has brought about new challenges, such as managing massive amounts of data and ensuring efficient training processes.

So, let's get into it.

## Some Key Terms

There are three terms we should clarify before going any further:

### *Foundation models*

Foundation models are advanced machine learning architectures that serve as the foundational building blocks for creating specialized models. They are pretrained on massive datasets, often consisting of text and recently including other data types like code, images, audio and video,, to develop general language-understanding and pattern-recognition capabilities. These models encode statistical relationships and linguistic structures from their training data, forming a robust starting point for further fine-tuning. This fine-tuning tailors the models to specific tasks or applications, such as powering large language models or other AI-driven solutions.

## *Large language models*

Large language models (LLMs) are specialized implementations of foundation models that have undergone additional training or fine-tuning to excel in specific language-based tasks. These models are designed to predict and generate human-like text by analyzing and emulating natural language patterns. LLMs are highly versatile, supporting several natural language processing (NLP) applications such as text generation, sentiment analysis, language translation, question answering, and more. Popular use cases include chatbots, content creation, multilingual communication, data analysis, code generation, recommendation systems, and virtual assistants. The “Use cases” section at the end of this chapter will look at these applications in more detail.

## *Generative AI models*

Generative AI refers to foundation models that were trained specifically to generate content based on the patterns and information they have learned. Generative AI models, and specifically LLMs, are known for their ability to generate coherent and contextually relevant text, images, audio and even video. In the context of LLMs, they can generate text responses, creative stories, product descriptions, and more, based on input and learned patterns.

Confusingly, these terms are frequently used interchangeably and loosely. For example, a popular image generation model, DALL-E, is better categorized as a generative AI model than as a large language model. Recently, however, the DALL-E image generation functionality has been integrated in the ChatGPT chatbot, one of the most popular large language model applications. Therefore, a user can ask a large language model (ChatGPT) to generate images. Over time, the language seems to be evolving towards calling all of these “AI models”, for simplicity.

## Transformer Models

The transformer model, introduced by the paper *Attention is All You Need* (Vaswani et al. 2017), marked one of the biggest shifts in how we approach sequence-based tasks. Transformers have set new standards in how to handle language data.

Transformers introduced major improvements over previous architectures by using self-attention and parallel processing, allowing them to handle sequences more efficiently and capture long-range dependencies effectively. *Self-attention* is a mechanism that allows each word or token in a sequence to focus on other words in the same sequence, regardless of their position. This is achieved by calculating a set of attention weights that measure the relevance of each token in the sequence to every other token. For instance, in a sentence, self-attention can help a word like *it* to align itself with its correct reference, even if that reference is several words away.

Before transformers, the most popular solution for natural language processing (NLP) tasks was Recurrent Neural Networks (RNNs). RNNs process data sequentially, one step at a time, which makes them suitable for handling time-dependent data like text. However, this sequential processing introduces a significant drawback: RNNs often struggle to retain information from earlier steps as they move forward in the sequence, especially over long inputs.

During neural network training, the model processes input data and generates predictions. These predictions are compared to the correct answers using a loss function, which calculates the error (how far the predictions are from the correct answers). An algorithm, such as *backpropagation*, calculates *gradients*: values that indicate how the model's parameters (weights and biases) should be adjusted to reduce the error and improve accuracy.

However, in long sequences like those handled by RNNs, gradients can become very small as they are repeatedly multiplied during backpropagation. Over time, these small values may shrink so much that computers treat them as zero, effectively stopping the model from learning.

This issue is known as the *vanishing gradient problem*, and it prevents the model from learning long-term dependencies in the data.

Transformers, on the other hand, overcome this limitation by using self-attention mechanisms. Instead of processing data one step at a time, transformers analyze all input tokens (e.g., words in a sentence) simultaneously. Self-attention is a mechanism that allows each word or token in a sequence to focus on other words in the same sequence, regardless of their position. This is achieved by calculating a set of attention weights that measure the relevance of each token in the sequence to every other token. For instance, in a sentence, self-attention can help a word like “it” to align itself with its correct reference, even if that reference is several words away.

Self-attention allows the model to weigh the importance of each token relative to others in the input, enabling it to capture relationships across the entire input sequence efficiently. This parallel processing not only speeds up computation but also eliminates the issues associated with sequential processing, like the vanishing gradient problem.

This enabled transformer-based models to excel in various NLP tasks, including translation, summarization, and question-answering, thanks to their ability to manage long-range dependencies and handle vast amounts of data. Their ability to focus on different parts of the sequence regardless of distance, along with positional encoding to retain sequence order, allows transformers to handle long sequences without losing context.

Now, some people wondered, well, since they can be scaled much better now, how about we throw more computing power and a lot more data at these models to see what happens? Models like GPT-3, LLaMa and their successors demonstrated that increasing the number of parameters can significantly improve the performance of transformer models.

Transformers have extended their influence beyond NLP into image processing with innovations like the Vision Transformer (ViT), which treats image patches as sequences and applies transformer models to them. ViT has shown promising results in image classification, offering a viable

alternative to the previous solution, convolutional neural networks (CNNs). Additionally, in recommender systems, transformers' ability to model complex patterns and dependencies enhances accuracy and personalization.

*Table 1-1. Table 1-1. The evolution of different neural network models from task to deployment ease.*

	<b>CNNs</b>	<b>RNNs</b>	<b>Transformers</b>
Application	More suited for spatial based tasks aka images	Well suited for sequence based tasks aka NLP.	Well suited for capturing all three modalities including images, NLP, and speech
Computation	Highly parallelizable input processing	Sequential processing	Parallel processing of inputs
Performance in Language Specific Tasks	Need large number of stacked convolution blocks for handling long-range dependencies	Can handle long-range dependencies much better than CNNs but can only handle the dependencies well to a given length	Can handle long to very long range dependencies much better than other architectures like RNNs or LSTMs
Scalability	Scalable	Limited scalability	Highly scalable
Data Requirements	Work well even on small datasets	Work well even on small datasets	Don't work well on small datasets

	<b>CNNs</b>	<b>RNNs</b>	<b>Transformers</b>
Ease of Training	Easy to train and tune	Require more tuning than CNNs	Difficulty to train and tune
Interpretability	Easy to debug	Difficult to debug	Difficult to debug
Deployment	Easy to deploy	Easy to deploy	Difficult to deploy
Small Edge Devices	Works well on edge devices	Works well on edge devices	Limited support for edge devices
Explainability	Supports wide variety of explainability	Limited explainability	Very limited explainability

This trend of throwing more compute and data at transformers is what sparked the evolution of LLMs, as well as the shift from an architecture that can do well on a single modality to one that generalizes on most modalities. Understanding this evolution can help you appreciate the differences in model architectures.

## Large Language Models

LLMs excel at understanding context and making associations between words, phrases, and concepts to provide relevant information based on the input query or prompt. While structured knowledge bases rely on human-curated data, LLMs can automatically extract knowledge from unstructured text. When trained on diverse textual sources, they can process a vast

amount of information without explicit human intervention. However, this also introduces a challenge, as the model can learn biased or incorrect information from the training data.

LLMs are also designed to understand and generate human-like text and to be accessible through natural language queries in conversational, interactive settings. This makes them convenient and user-friendly for retrieving information and obtaining responses.

These models are “large” not just because of the amount of data they’re trained on, but also because of their number of parameters. In neural networks, *parameters* are weights and biases. When an input like a prompt is presented to a model, it first transforms the input into a numerical representation, and then the numbers are processed through the neural network. Each node in the neural network contains a bias, adding or subtracting to the input value, and each connection between nodes contains a weight that will multiply the value of the input as it passes through nodes. Using more parameters greatly extends the capabilities of traditional transformer models, but not without massive tradeoffs in cost and evaluation complexity.

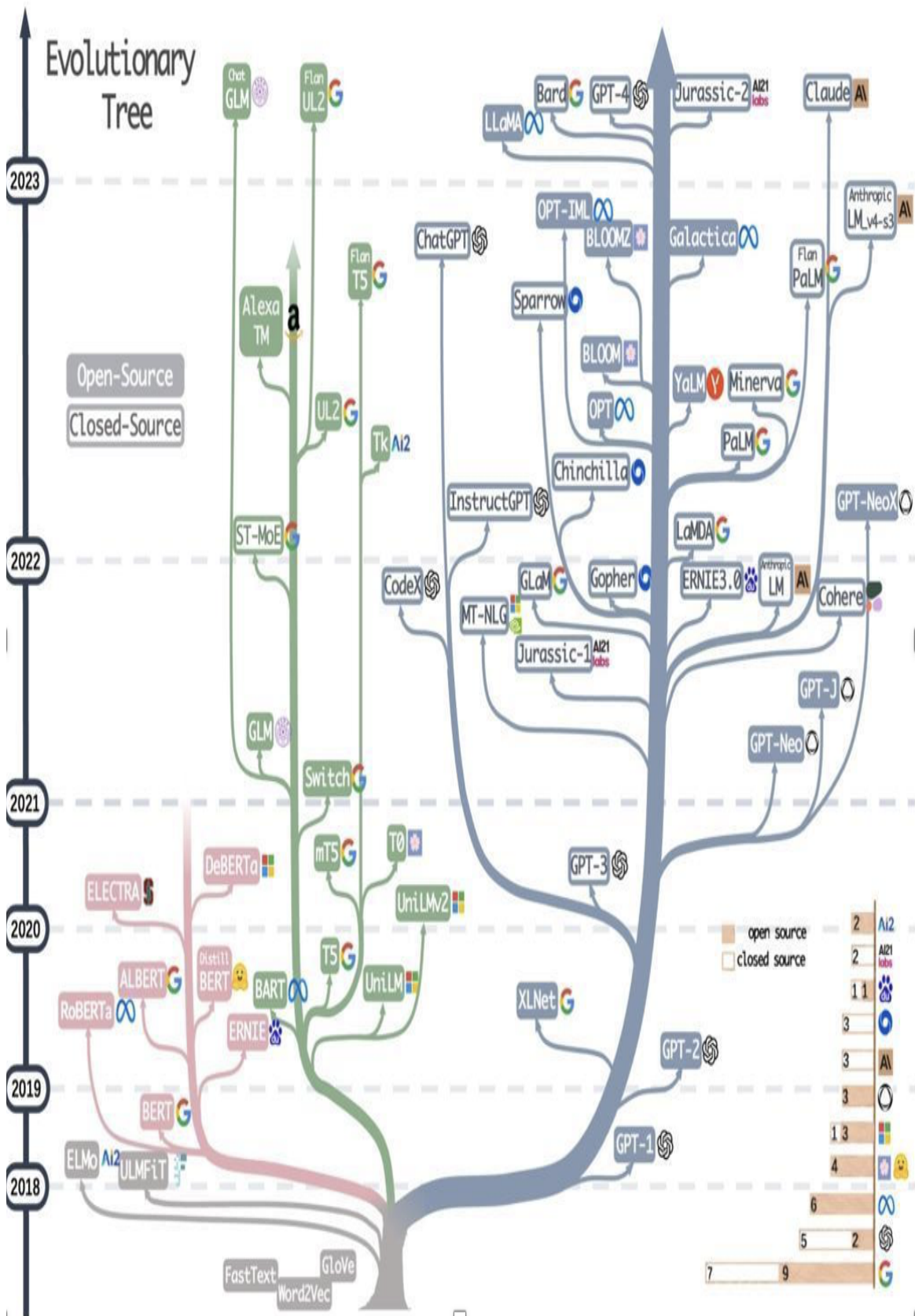
There are two basic categories of LLMs, discriminative and generative. *Discriminative* models, such as BERT (Bidirectional Encoder Representations from Transformers), which was introduced in 2018, learn the boundary between classes in a classification problem. They’re concerned with the conditional probability  $P(y | x)$ , which is the probability of the output given the input. Discriminative transformer models are typically used for tasks like text classification, sentiment analysis, and named-entity recognition, where the goal is to predict a label or category given some input text.

Generative models, such as GPT-3 and GPT-4, learn the joint probability distribution of the input and the output, or  $P(x, y)$ , and can generate new data points similar to the training data. Generative models are used for tasks like text generation, where the goal is to generate new text similar to the text the model was trained on. Not all LLMs need to be generative,

although most are. Throughout this book, when we refer to “LLMs,” we mean generative LLMs.

## **LLM Architectures**

There are two main types of architecture for language models: encoders and decoders. Encoders and decoders can also be combined, and there is ongoing research on new architectures. We discuss the details below:



## Encoder-Only LLMs

*Encoder-only models* are designed to process and comprehend input text, transforming it into a meaningful representation or embedding. Embeddings are numerical representations of data, such as words, phrases, or sentences, in a high-dimensional vector space. These embeddings capture meaning and context in a way that results in words with similar meanings or contexts to be placed close together in this vector space. This representation captures the essence of the input, making it suitable for tasks where understanding the context is needed.

One of the most notable examples of an encoder-only model is BERT (Devlin et al. 2019). During its pretraining phase, BERT uses *masked language modeling*, a technique where random words in the text are masked and the model learns to predict these masked words based on the surrounding context. BERT is also trained on the next-sentence prediction, where it determines if one sentence follows another logically.

The primary advantage of encoder-only models lies in their syntactic understanding of text: capturing the intricate relationships between words and their contexts. These models excel in tasks such as sentiment analysis, named entity recognition, and question answering.

However, encoder-only models have their limitations. They are not designed for generating new text; their focus is solely on understanding and analyzing the input. This limitation can be restrictive when using them in applications requiring text generation or completion.

## Decoder-Only LLMs

*Decoder-only models* are good at generating coherent and contextually relevant text based on an input or prompt. The Generative Pre-trained Transformer (GPT) series, including GPT-2, GPT-3, and the more recent GPT-4, are examples of this architecture.

These models are pretrained using a *language modeling objective*, a technique where they learn to predict the next word in a sequence given the preceding context. This training approach allows them to generate text that flows naturally and maintains coherence over longer passages.

The key advantage of decoder-only models is their ability to generate high-quality text. This makes them highly effective for tasks such as text completion, summarization, and creative writing. They also exhibit *emergent properties*, meaning that they can perform tasks beyond their initial training objective, such as translation and question answering, without additional fine-tuning.

However, their focus on text generation can be a limitation in tasks requiring deep understanding of the input text. Decoder-only models generate text based on patterns learned during training, which may not always align with the specific nuances of the input.

## Encoder-Decoder LLMs

*Encoder-decoder models* combine the strengths of both encoder and decoder architectures, making them suitable for tasks involving complex mappings between input and output sequences.

In this setup, the encoder processes the input text to create an embedding, which the decoder then uses to generate the output text. Notable examples include BART (Bidirectional and Auto-Regressive Transformers) and T5 (Text-To-Text Transfer Transformer). BART, introduced in 2019, is trained using *denoising auto-encoding*, where parts of the input text are corrupted and the model learns to reconstruct the original text.

The encoder-decoder architecture excels in tasks where the input and output are different in structure and length, such as machine translation and text summarization. However, the complexity of training and the computational resources they require for encoder-decoder models can be a drawback. Their dual architecture means they must effectively integrate both components, which can be demanding in terms of both data and processing power.

## State Space Architectures

A new approach tries to solve one of the problems with transformers, which is that the self-attention mechanism has *quadratic complexity*, since the relationship between each pair of tokens needs to be modeled. Quadratic complexity is generally a hard computational problem, especially when using larger datasets.

The state space architecture replaces the transformer approach by incorporating *state space representations*, which model the state of the system, instead of recording it at each step. This compression allows for linear computational complexity, improving computational performance and reducing memory requirements, but increases the errors.

Researchers are trying to solve the error problem. Recent examples are Mamba (Dao and Gu 2024) and Mamba-2 (Dao and Gu 2024), which try to solve the error problem by creating a state representation that dynamically attempts to determine the important parts of the prompt by modeling importance as a state space parameter. In experimental settings, Mamba performs as well as transformers-based models that have double the number of parameters for small and medium prompts but still has not delivered on the promises of low error rates for larger prompts.

## Architecture strengths and weaknesses

LLM architectural designs each have their own sets of strengths and limitations. Encoder-only models like BERT are highly effective for understanding and analyzing text but fall short in generating new content. Decoder-only models, exemplified by the GPT series, excel in generating coherent and contextually relevant text but are non-deterministic, which can be problematic for some applications like text classification. Emerging architectures like state space models promise enhancements in performance and applicability. They should be monitored, but haven't been proven yet.

## Small Language Models

Another recent development is *small language models* (SLMs): compact, efficient language models designed to perform NLP tasks while using fewer computational resources than LLMs.

Unlike LLMs, which contain billions of parameters and require substantial memory and processing power, SLMs are often designed to have millions or even just hundreds of thousands of parameters. The tradeoff is that they must focus on specific tasks or subjects. This makes them lightweight, cost-effective, and deployable on a wider range of devices, including mobile phones and IoT edge devices, and in environments with limited computational resources. The development of SLMs has been driven by the demand for efficient, accessible AI solutions that can operate in real-time and offline, providing functionality without relying on cloud-based infrastructure.

SLMs do not perform well in tasks that require contextual understanding, extensive memory, or reasoning abilities. They are not intended for more general problem-solving and need to be fine-tuned on specific datasets to perform well on particular tasks, maximizing efficiency while maintaining accuracy within a defined scope. While LLMs tend to perform several NLP tasks reasonably well in a large number of domains, SLMs need to be specifically trained. For instance, an LLM might be able to perform moderately well at summarizing legal documents as well as medical articles, while an SLM would excel in one and perform poorly at the other.

## Choosing an LLM

In the LLM world, it's easy to get swept up in the excitement of the latest breakthroughs and cutting-edge technologies. New models pop up all the time. The truth is, selecting the right LLM is more than just a technical decision; it's a strategic choice with far-reaching implications. Here are five reasons why the model you choose can make all the difference:

*Alignment with objectives*

Are you looking for a model that excels at generating human-like text? Or do you need one that can understand complex queries and provide accurate responses? The specific capabilities of different models can vary significantly. Some are designed with a focus on conversational abilities, while others are optimized for tasks like summarization or translation. Choosing a model that aligns with your objectives ensures that you're investing in a tool that will deliver the results you need.

### *Performance and efficiency*

Not all LLMs are created equal. Larger models might offer impressive performance and efficiency, but often come with high computational costs and slower response times. Smaller, more optimized models tend to provide faster results and be more cost-effective, but rarely match the performance of their larger counterparts.

### *Training data and bias*

The training data used to develop an LLM shape its behavior and outputs. Variations in the dataset on which models are trained can lead to variations in how they handle specific topics or issues. Some models exhibit biases based on their training data, which can impact the accuracy and fairness of their responses. Choosing a model with a diverse and representative training dataset can help mitigate these risks and ensure more reliable and equitable outcomes.

### *Customization and adaptability*

Your needs might not fit neatly into the one-size-fits-all approach of a generic LLM. Some models offer greater flexibility and can be fine-tuned or customized to better suit your specific requirements. If that's what you need, choose one with strong customization capabilities so that you can mold it to better fit your use case.

### *Integration and support*

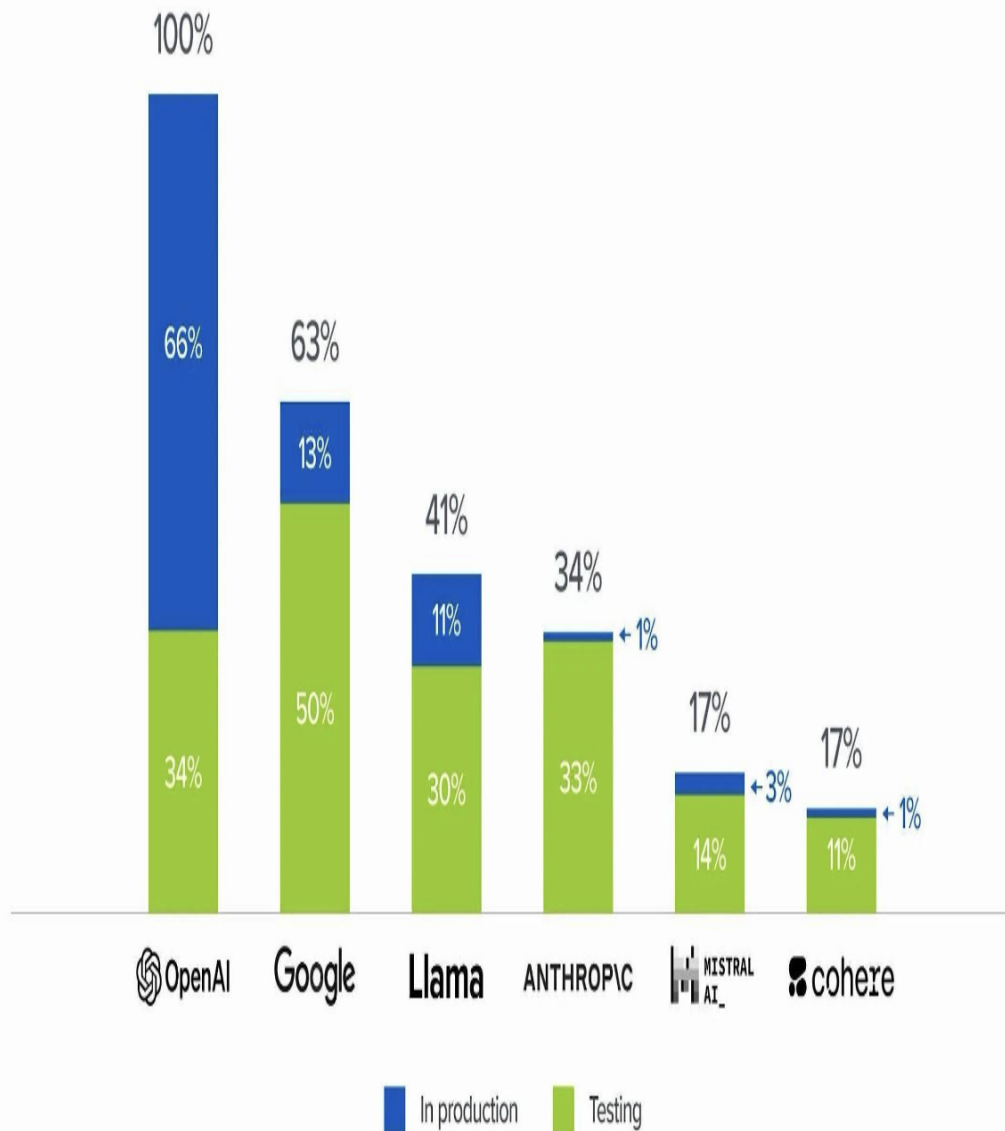
The practical aspects of integrating an LLM into your existing systems and workflows cannot be overlooked. Some models come with robust support and documentation, making integration smoother and less time-consuming. Others require more effort to set up and maintain. Considering how well a model integrates with your infrastructure and the level of support available can save you time and reduce headaches in the long run.

Overall, the LLM model you choose is not just a technical decision; it's a strategic one that impacts the effectiveness, efficiency, and overall success of your AI initiatives. Remember: the model you choose matters. By carefully evaluating your needs and understanding the strengths and limitations of different models, you can make an informed choice that aligns with your goals and sets you up for success.

## **The Big Debate: Open-Source vs. Proprietary LLMs**

Companies must navigate a complex landscape when choosing between open-source and closed-source, and open-weight LLMs. This section looks at each option's limitations and benefits.

## Which model providers are enterprises using?



Source: a16z survey of 70 enterprise AI decision makers

Figure 1-2. Enterprise adoption of different proprietary LLMs.

Open-source and open-weight are two types of publicly accessible LLMs that have gained traction in the AI community as of this writing, particularly among those looking to customize, deploy, or study advanced AI without relying on proprietary solutions.

*Open-source* LLMs are models with freely available underlying source code. Anyone can inspect, modify, and potentially redistribute the model and its architecture. These models typically include details about the architecture, training methods, and source code for the framework. Using open-source models provides technical transparency and adaptability, and fosters a community of collaboration. However, open-source LLMs may or may not come with pretrained *weights*, the trained parameters that make the model functional and useful for specific tasks. These weights are the model's "knowledge" gained from its training on large datasets and are essential for the model to perform effectively without retraining from scratch. Companies that want to take advantage of such models may need to acquire the training data themselves.

With *open-weight* LLMs, the weights are publicly accessible. Having access to open weights means that users can directly deploy the model for real-world applications like text generation, summarization, and translation, or fine-tune it on their own data. While many open-weight models are also open-source, some restrict use for commercial applications or require adherence to specific licensing terms, as seen with models like Meta's LLaMA series.

The distinction between open-source and open-weight LLMs is crucial in determining how accessible and useful a model is "out of the box." Open-source models without weights can still allow for architectural experimentation and model-training setups, but they lack immediate functionality for practical applications until they are trained, which requires substantial computational resources. In contrast, open-weight models provide ready-to-use capabilities, making them more accessible to developers who do not have the resources for large-scale model training but want to fine-tune or deploy a pretrained model.

By leveraging open-source and/or open-weight models such as Llama or Mistral, companies can deploy models on existing hardware, which can be more cost-effective than cloud-based proprietary solutions, which involve renting hardware. This can be particularly advantageous for startups or small to medium-sized enterprises (SMEs) operating under tight budget constraints. For these companies, the financial savings can now free up resources for other aspects like fine-tuning.

Besides financial concerns, a company may have additional requirements, such as wanting to ensure that the training data includes or excludes specific datasets. In these cases, an open-weights model is not sufficient; you really need an open-source model. For example, if your company wants to guarantee that a model has never seen a specific data point, an open-weights model whose training dataset is not shared can't offer such a guarantee.

Community support is another potential advantage for open-source LLMs. The collaborative nature of the open-source ecosystem means that developers, researchers, and organizations continuously contribute to improving these models, and newly fine-tuned models are easily available via Hugging Face. Companies benefit not only from this collective intelligence, but from access to a wider range of resources, tools, and best practices. This community-driven development is dynamic and evolving, and is often where new developments begin.

However, the open-source/open-weights approach is not without its challenges. Maintenance and support can be significant hurdles. Data privacy and security also emerge as big concerns. Transparency can be a double-edged sword, exposing a company to potential risks even as it demands significant effort to safeguard sensitive information and comply with data-protection regulations. Ensuring that these models do not become a vector for security breaches requires meticulous attention and proactive measures.

Scalability and performance are additional considerations. Open-source LLMs aren't always optimized for large-scale deployments. Companies

with substantial operational demands might face performance bottlenecks or scalability challenges. The customization required to adapt open-source models for enterprise-grade applications can be resource-intensive and require significant engineering efforts.

Open-source and open-weight language models also introduce security concerns. Anyone can immediately use, fine-tune, or modify pretrained open-weight models and potentially apply them in harmful ways, such as generating misinformation, creating realistic fake content, or deploying automated tools for phishing and social engineering. Since open-weight models' training data often includes both public and proprietary datasets, they can also sometimes unintentionally generate or reveal sensitive or biased information embedded in the training data, posing privacy risks.

Furthermore, open-source models, which include the code and architectural blueprints, are vulnerable to manipulation and exploitation. Malicious actors can introduce harmful code or adjust models to bypass safety mechanisms, then distribute these altered versions under the guise of legitimate software. This can lead to scenarios where organizations unknowingly adopt models that include backdoors or biased, harmful outputs. The decentralized nature of open-source development means that code modifications don't always go through rigorous security checks, leaving room for vulnerabilities that could be exploited. Addressing these security challenges requires adopting responsible AI practices, including rigorous code reviews, security audits, and clear usage policies to mitigate risks while promoting open collaboration.

Carefully review any contractual restrictions on usage before adopting a model. You don't want to build a whole commercial application around an open-source LLM only to find out that the open-source LLM does not allow commercial usage.

On the other side of the spectrum are *closed-source*, or *proprietary*, LLMs, such as those developed by leading tech giants. These models often come with robust support and maintenance, including dedicated assistance for troubleshooting and optimizing performance. This support infrastructure

ensures that any issues encountered are addressed promptly, allowing companies to focus on their core activities without being sidetracked by technical difficulties.

Company	Model	Input	Output	Tokens
OpenAI	GPT-4 Turbo	\$0.01	\$0.03	1000
OpenAI	GPT-4	\$0.03	\$0.06	1000
OpenAI	GPT-3 Turbo	\$0.001	\$0.002	1000
Google	Gemini Pro	\$0.00025	\$0.0005	1000
Anthropic	Claude Instant	\$0.0008	\$0.0024	1000
Anthropic	Claude 2.1	\$0.008	\$0.024	1000
Cohere	Command Light	\$0.0003	\$0.0006	1000
Cohere	Command	\$0.001	\$0.002	1000
Mistral AI	Mistral Tiny	\$0.00015	\$0.00046	1000
Mistral AI	Mistral Small	\$0.00066	\$0.00197	1000
Mistral AI	Mistral Medium	\$0.00274	\$0.00821	1000

Figure 1-3. Pricing Comparisons (as of 2024) <https://subpage.app/blog/2024-Commercial-AI-LLM-pricing-compared-in-detail-GPT-Gemini-Cohere-Mistral>

Closed-source LLMs are generally optimized for large-scale deployments, making sure that they can scale with operational loads effectively, so they often come with performance guarantees. Their performance benchmarks often reflect their ability to deliver consistent and reliable results, which is a critical factor for companies with high operational demands.

One of the primary limitations of the closed-source approach is the high cost. Another is the lack of transparency: companies have limited visibility into the internal workings of these models. While this concern may seem unusual, consider a scenario in which a commercial LLM provider inadvertently consumes private data during training. You use this LLM provider for your own application, and some of your users realize how to get your application to reveal the private data. The people whose data was revealed sue you. We recommend that you fully understand what legal protections are in place when using information services like third-party LLMs.

Regardless of these drawbacks, companies are right now willing to make expensive bets hoping for excellent returns in the future from investing in GenAI applications.

[lucas.meyer@gmail.com](mailto:lucas.meyer@gmail.com)

## Enterprise Applications for LLMs

LLMs are transforming enterprise operations in many industries, from changing how we retrieve knowledge to enhancing autonomous agents. This comes down to several applications like knowledge retrieval, translation, audio-speech synthesis, recommender systems, and autonomous agents.

### Knowledge Retrieval

People have long used search engines to discover information, but their limitations have become more apparent as data volumes and complexity grow. LLMs offer a new paradigm for accessing and using information.

Unlike conventional systems, which rely heavily on keyword matching and ranking algorithms, LLMs bring a conversational, personalized approach to information retrieval.

Users can engage in long conversations with LLMs. Instead of simply receiving a list of links or documents, they can set parameters for the tone, intent, and structure of the information they need. This capability transforms the search experience from a transactional process into a dynamic dialogue. For example, an LLM can interpret a request like, “Explain this concept as if I were a beginner,” and provide a tailored explanation that’s both accessible and relevant.

On the data retrieval side, LLMs can enhance productivity tools, for example through integrations with office software suites like those from Google and Microsoft. Imagine querying a spreadsheet with natural language to extract insights or asking a document to summarize key points. This simplifies data management and makes complex information more accessible. Furthermore, LLMs can integrate with internal systems to automate routine tasks and create knowledge graphs, streamlining workflows and enhancing organizational efficiency. However, while LLMs improve the accuracy and relevance of information retrieval, they also require meticulous handling to ensure data privacy and system security.

## Translation

Translation is another domain where LLMs are being used heavily. Traditional machine-translation systems often struggled with languages that had limited datasets, relying on statistical methods. LLMs are changing this by offering zero-shot and few-shot translation capabilities. *Zero-shot* refers to the model’s ability to translate languages without prior examples, a feat that was previously challenging. *Few-shot*, on the other hand, allows LLMs to perform well with minimal data.

This is particularly advantageous for translating languages that are underrepresented in training datasets. For companies involved in global operations or content creation, this is a major selling point. It eases

localization of content, such as subtitling films or translating marketing materials, without extensive data requirements, allowing companies to expand into new markets without investing too many resources upfront.

LLMs trained on multilingual datasets can easily adapt to new languages, allowing translations across a broader spectrum of languages, including those with sparse resources. The applications for this extend to literature, film, and even real-time communication, where accurate and contextually appropriate translation can be helpful.

Yet, while LLMs offer significant improvements over traditional translation methods, maintaining accuracy and handling idioms still remain open challenges.

## **Speech Synthesis**

The ability to generate speech that resonates with human listeners can significantly enhance user experience and interaction. *Speech synthesis*, generating audio that mimics human speech from text, is another area where LLMs are making remarkable progress. Historically, speech synthesis systems have struggled with creating natural and engaging audio outputs: the sound generated sounded clearly “robotic”. LLMs, however, have the potential to revolutionize this field by generating human-like speech with impressive fidelity. With training on text and audio datasets, LLMs can understand and replicate the subtleties of human speech, such as intonation, rhythm, and stress.

This is useful for applications like virtual assistants, realistic voiceovers for characters in video games, or engaging audio content for educational materials. However, audio-speech synthesis still has room to improve, especially with regard to recognizing accents and other variations in speech.

Using LLMs to automate the creation of speech content makes it easy for businesses to produce large volumes of content without the time and costs of extensive manual recording.

## Recommender Systems

Recommender systems are at the heart of many digital platforms, from ecommerce to streaming services. LLMs enhance these systems by incorporating a deeper understanding of users' preferences and contextual factors. Earlier recommender systems relied on historical user data and predefined algorithms, which often led to limited or repetitive suggestions. LLMs, with their ability to process and interpret diverse data sources, offer a more nuanced approach.

LLM-powered recommender systems can analyze user interactions, preferences, and even conversational cues, including audio and video inputs, to deliver personalized recommendations in real time. For example, if a user describes a product in natural language and provides an image, the LLM can integrate both modalities to offer more relevant suggestions, even in response to ambiguous or vague requests.

Despite these advantages, many challenges still remain unsolved. Maintaining user trust requires careful attention to the model's transparency and reasoning.

## Autonomous AI Agents

*AI agents* are designed to perform specific tasks autonomously, leveraging LLMs to execute complex operations that would otherwise require human intervention.

For example, in a customer service environment traditional automated agent systems might follow rigid scripts or rely on basic rule-based logic. LLM-powered AI agents, however, can engage in dynamic, context-aware conversations. They understand user queries more deeply, interpret intent more accurately, and generate responses that are more natural and engaging.

In project management, LLMs can power intelligent project assistants that manage schedules, set reminders, and even draft project reports. These AI agents can interact with team members, understand project requirements, and adapt their responses to ongoing developments.

## **Agentic Systems**

Agentic systems represent a more novel application of LLMs, where AI agents not only perform tasks but also make strategic decisions. These systems leverage LLMs' data processing and analysis capabilities to discern patterns and make informed decisions in real time. This is particularly helpful in environments where decisions need to be based on complex, multifaceted information (as shown by the example workflow in Figure 1-4).

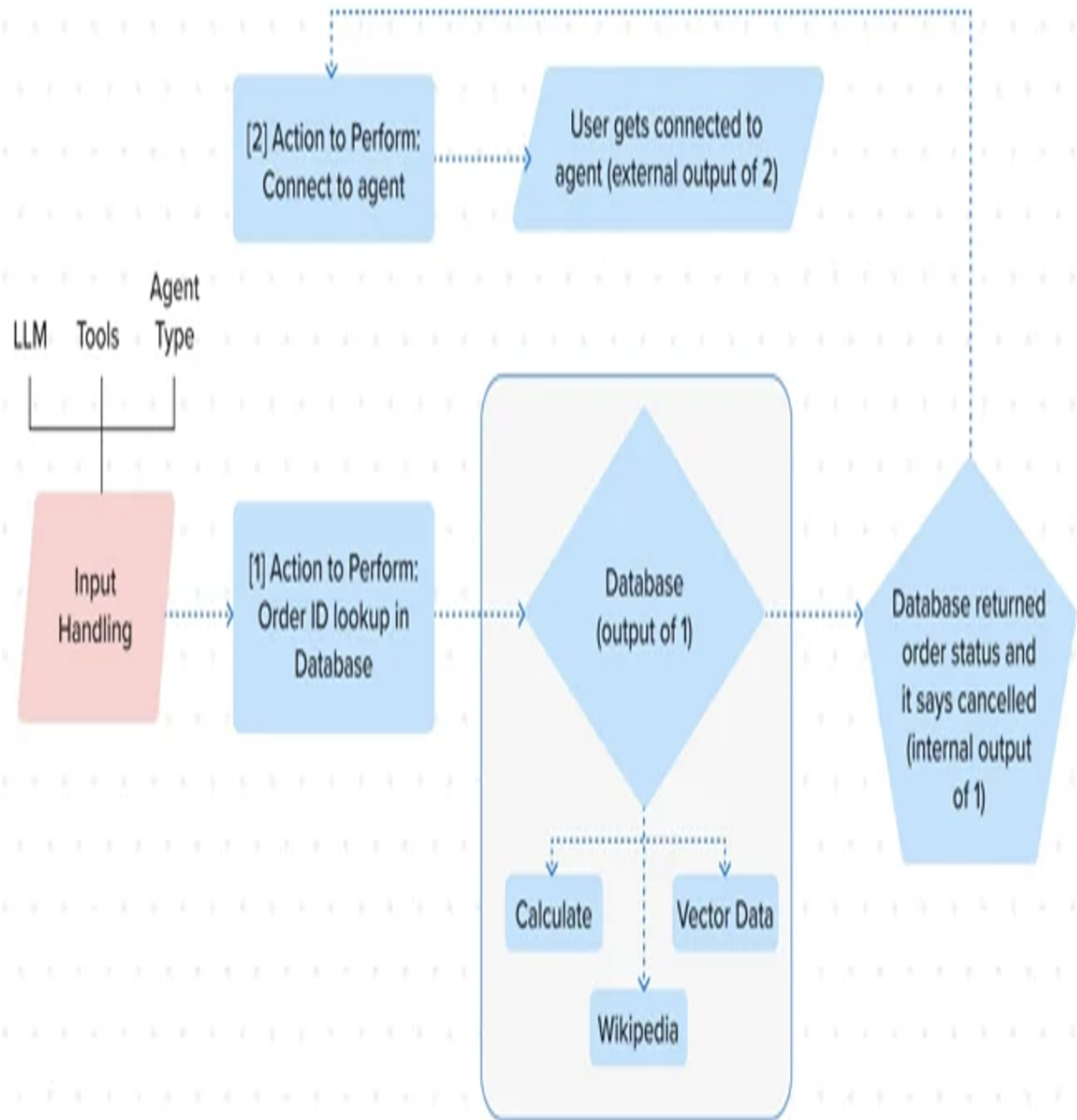


Figure 1-4. Agentic AI in the enterprise. Source: [https://www.haptik.ai/knowledge-center/agentic\\_ai](https://www.haptik.ai/knowledge-center/agentic_ai)

In finance, agentic systems can digest data from financial reports, news articles, and market analytics, then use it to analyze market trends, assess risk factors, and make investment recommendations that align with investment strategies.

Similarly, in supply chain management, agentic systems can optimize inventory levels, predict demand fluctuations, and coordinate logistics based on data from various sources—such as sales forecasts, supply chain disruptions, and production schedules.

However, these systems aren't always reliable. Integrating them into existing workflows requires careful planning. Companies must consider how AI agents and agentic systems will interact with human teams, how they will be managed, and how their outputs will be monitored. Clear guidelines and oversight mechanisms are essential to ensure that these systems complement rather than disrupt existing operations. We discuss these issues in Chapter 8.

Data security and privacy are also big concerns. LLMs handle vast amounts of sensitive information, and protecting it from breaches or misuse is key. You need to establish strong data governance policies and invest in security measures to safeguard against potential risks. We also discuss these issues in Chapter 8.

## **Ten Challenges of Building with LLMs**

LLMs introduce several new challenges, which can be amplified by their enormous scale and the numerous applications that they can be used in. Addressing these challenges is important for integrating and deploying LLMs in production. We list ten challenges below, and point to the book chapters where we address them.

### **Size and Complexity**

LLMs generally have millions or even billions of parameters. This makes training, monitoring, and evaluating them extremely complex. Moreover, being generative models, they can fail silently, causing hallucinations and inaccurate information. Addressing this requires a structured approach that includes benchmarks used for machine learning but adds several other techniques, which we discuss in Chapter 7.

### **Training Scale and Duration**

Training LLMs requires processing large datasets. This is not only difficult from the data management perspective, but also memory and the

computational resources required for training the models. We discuss this in Chapter 3.

Training LLMs can take days, weeks, or even months, and managing parallel and distributed training across large clusters of GPUs and TPUs requires specialized hardware and organizational skills. This means that hardware represents a major dependency on external organizations and market availability, and requires careful, systematic planning. We discuss this in Chapter 9.

Handling large, potentially sensitive training datasets requires careful security measures and anonymization, which we discuss in Chapter 2.

## Prompt Engineering

One of the most common ways of making an LLM work better for a specific problem is *prompt engineering*, the science and art of crafting the text inputs that are sent to the models. Prompt updates can significantly improve or degrade the user experience. But prompt engineering is iterative and can be difficult to master and document, especially with closed-source LLMs. We discuss this in Chapter 5.

Updates on the proprietary models like OpenAI's GPT-4 can result in significant *model drift*, where the same inputs suddenly provide a different output due to a model update, requiring more effort and financial commitment to fix. This becomes additionally complex when you have orchestration frameworks. If your infrastructure relies heavily on prompt-engineering pipelines, monitoring is crucial, and we discuss it in Chapter 7.

## Inference Latency and Throughput

Responses provided by LLMs are also called *inferences*. LLMs are often deployed in applications that require real-time or near-real-time responses, which means optimizing for speed becomes important. This can be especially complex in dynamic models like LLMs. Also, maintaining high throughput without having access to model parameters can add to additional complexity for LLMOps teams. Edge devices used in IoT applications

introduce even more challenges related to limited computational resources and varying network conditions.

## **Ethical Considerations**

Like any machine learning model, LLMs generate outputs based on the data that they have been trained on. LLMs applications are frequently designed to seem like the user is chatting with a human instead of a machine, making them accessible to a much larger user base than specialized machine learning systems and greatly increasing the impact of potential biases introduced by the training data.

We discuss techniques for monitoring the outputs in Chapter 7, and discuss the privacy and ethical implications in Chapter 8.

## **Resource Scaling and Orchestration**

The scale at which LLMs operate often requires load balancing and dynamic resource scaling. Different proprietary models can also behave very differently based on the use case, and constant scenario modeling is expensive and time-intensive. We discuss how to manage dependencies across various components in distributed multi-model environments to ensure reliability and scalability in Chapter 5.

## **Integrations and Toolkits**

LLMs require several new integrations and toolkits that are adapted to both generative as well as discriminative use-cases, involving communicating with various APIs. Integrating these LLMs into existing systems requires robust security protocols to prevent vulnerabilities and potential misuse. Changes in LLMs and version management can also lead to compatibility issues across the stack. We discuss these in Chapter 4.

## Broad applicability

LLMs are adaptable and easy to use, which means that they can be applied to numerous consumer-facing applications, as we will see in Chapter 6. This makes them more likely to be exposed to untested scenarios than traditional machine learning systems, and require a faster feedback loop to monitor and improve their performance. We address techniques for monitoring in Chapter 7.

## Privacy and Security

Collecting real-time information involves handling user data, sometimes including personally identifying information (PII). This means security and privacy become the cornerstone of maintaining trust and regulatory compliance. This challenge extends well beyond inference monitoring, touching the domain of cybersecurity.

Even companies like OpenAI have received reports about database leaks (OpenAI 2023) into user accounts that made chat interactions visible to unauthorized users. We talk more about Privacy and Security in Chapter 8.

Regularly auditing your data management processes, both internally and externally, is also vital for enhancing user trust and complying with legal requirements. We discuss best practices for data management in Chapter 3.

## Costs

One of the biggest considerations for LLMs is cost, both immediate and long-term. While most transformer models require expensive training, maintaining and scaling LLMs incurs the highest costs, especially in the inference stages. You could end up paying even for failed requests, so experimenting with model performance can become very expensive very quickly for companies building on closed and proprietary models.

Even in open-source models, excessive fine-tuning can quickly lead to a phenomenon called *overfitting*, where the model appears to perform extremely well because it learns the training dataset, but does not generalize

to the unseen data that will be presented to it by real users. There are always tradeoffs between generalization ability and cost, which we explore in Chapter 5.

## Conclusion

Adopting LLMs requires careful consideration and strategic planning to navigate these intricate challenges, which requires a new discipline and a set of new tools to succeed. We call this discipline LLMOps, and we start our journey by defining it in the next chapter.

## References

Dao, Tri, and Albert Gu. 2024. “Transformers Are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality.” arXiv. <https://doi.org/10.48550/arXiv.2405.21060>.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. “BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding.” arXiv. <https://doi.org/10.48550/arXiv.1810.04805>.

OpenAI. 2023. “March 20 ChatGPT Outage: Here’s What Happened.” March 24, 2023. <https://openai.com/index/march-20-chatgpt-outage/>.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. “Attention Is All You Need.” In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000–6010. NIPS’17. Red Hook, NY, USA: Curran Associates Inc.

# Chapter 2. Introduction to LLMOps

---

## A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book. The GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at [sgrey@oreilly.com](mailto:sgrey@oreilly.com).

The size and complexity of LLMs’ architecture can make productionizing these models incredibly hard. *Productionizing* means not just deploying a model but also monitoring it, evaluating it, and optimizing its performance.

Depending on your application, there are constantly new challenges, including how to process data, how to store and dynamically adapt prompts, how to monitor user interaction, and most pressing, how to prevent the model from spreading misinformation or memorizing training data (which can lead it to release personal information). That’s why *operationalizing* LLMs, which means managing them day-to-day in production, requires a new framework.

*LLMOps*, as it’s called, is an operational framework for putting LLM applications in production. Although its name and principles are inspired by its older siblings, MLOps and DevOps, LLMOps is significantly more nuanced. The LLMOps framework can help companies reduce technical

debt, maintain compliance, deal with LLMs' dynamic and experimental nature, and minimize operational and reputational risk by avoiding common pitfalls.

This chapter starts by discussing what LLMOps is and how and where it departs from MLOps. We'll then introduce you to the LLMOps engineer role and where it fits into existing ML teams. From there, we'll look at how to measure LLMOps readiness within teams, assess your organization's LLMOps maturity, and identify key KPIs for measuring success. Towards the end of this chapter, we will outline some challenges that are specific to productionizing LLM applications.

## What Are Operational Frameworks?

*Operational frameworks* provide a structured approach to managing complex workflows and pipelines within an organization. These frameworks integrate tools and practices to automate and streamline organizational processes and ensure consistency and quality across the project lifecycle.

Some of the earliest operational frameworks can be traced back to military strategy and the Industrial Revolution. Two of the most popular ones, both introduced in 1986, are Toyota's Lean Production System, which put Toyota ahead of most of its contemporaries, and Six Sigma, Motorola's data-driven approach to improving processes and reducing defects.

In 2008, the tech industry began adopting what is now one of the most popular operational frameworks in software: DevOps. (The term combines *software development* and *operations*.) In 2018, MLOps, an operational framework for nongenerative machine learning (ML) models, became the talk of the town; since then we've seen SecOps (Security Operations), DataSecOps, and many more.

With the massive adoption of LLMs in 2023, a new operational framework started floating around within companies building LLM applications:

LLMOps. LLMOps is still in its infancy, but as generative models become integral to software products, its popularity is likely to boom.

Figure 2-1 shows the slow rise of Ops frameworks over the years. LLMOps started picking up with the mass adoption of LLMs in early 2023. As of this writing, more and more enterprises are realizing that they can add value and profit with LLM-based offerings, leading to an upward trend for LLMOps. In fact, 2025 may become one of the first peaks for LLMOps frameworks.



IMAGE TO COME

*Figure 2-1. Google Ngram showing the rise in popularity of the term LLMOps from 2019 to 2023.*

Deploying LLMs can be as simple as integrating a chatbot into your website via an API, or as complicated as building your own LLM and frontend from scratch. But maintaining them to be performant (also called *productionizing*)—that is, keeping them reliable, scalable, secure and robust—is a massive challenge. This book, like LLMOps, is focused on exactly that: What happens *after* you deploy your LLM in production? In the upcoming sections, we will discuss it all.

## **From MLOps to LLMOps: Why do we need a new framework?**

There is some overlap between MLOps and LLMOps: both deal with the operational lifecycles of ML models, after all. They also share common principles in terms of managing ML workflows. However, the two frameworks diverge in their primary focuses and objectives, and while MLOps handles nongenerative models (both language and computer vision), LLMOps deals with generative language models—and thus with

mammoth levels of complexity. The complexity of these models owes not only to their scale and architecture, but also the unique processes involved in data engineering, domain adaptation, evaluation and monitoring for them. The key distinctions are apparent in LLMs' prediction transparency, latency, and memory and computational requirements.

Perhaps the biggest difference is the shift in how end users consume these models. Nongenerative ML models are predictive tools used for passive consumption, such as in dashboarding, recommendations, and analytics. By contrast, LLM applications are deployed as **Software 3.0** in consumer-facing applications for active user interaction. This brings several challenges from Software 1.0 (DevOps) back to the surface. In fact, it wouldn't be wrong to say that LLMOps shares more similarities with DevOps than with MLOps.

Building your own generative AI application requires tools, frameworks, and expectations to match the scale and complexity of these models, which is far beyond the scope of the existing MLOps solutions. To help you understand the differences between the various kinds of Ops frameworks, let's do a thought experiment.

Imagine MLOps as being something like building a small home from the ground up. In comparison, DevOps, which deals with the entire product lifecycle, would be like developing a large shopping complex. And LLMOps? It's more like building the Burj Khalifa. For all three frameworks, you are working with the same construction materials: wood, steel, concrete, bricks, hammers, and so on. Much of the basic process is the same, too: you lay down the foundations, lay down the plumbing, and finally build the walls. But you wouldn't contract your local construction workers to engineer the Burj Khalifa, would you?

Most of MLOps for natural-language modeling is based on building smaller, discriminative models for tasks like sentiment analysis, topic modeling, and summarization. For these tasks, you first hypothesize ideal features, model your data as a function of those features, and then optimize the model for that specific task.

LLMs, by contrast, are generative, domain-agnostic, and task-agnostic. That fundamental difference means you can use the same model for summarizing or for answering questions, without having to fine-tune it.

The best use cases for LLMs are when you don't know what features to optimize for (or, even better, if the features are too abstract) and when you need to model multimodal data within a single pipeline. Unlike smaller discriminative models, LLMs don't rely on predefined features or task-specific architectures. Instead, as you learned in Chapter 1, they are trained on vast amounts of text data to learn the patterns and structures in language itself. For LLMs, the training process involves a *loss function*, which measures how well the model's generated outputs match the expected outputs across various tasks. For example, during training, the model might generate a sequence of text; the loss function calculates the difference between this generated sequence and the target sequence.

Additionally, for a standard comparison, the hyperparameter space for a 175-billion parameter GPT-4 model would likely be approximately 1,500 times larger than a standard discriminative BERT model (which has 110 million parameters). This makes it incredibly costly to fine-tune and do the kind of iterative training that is typical in MLOps.

Table 2-1 outlines some of the key differences between the MLOps and LLMOps model lifecycles.

*Table 2-1. Comparing the MLOps and LLMOps model lifecycles*

<b>Lifecycle Step</b>	<b>Context</b>	<b>MLOps</b>	<b>LLMOps</b>
Data Collection	Ease	Always and is straightforward	Sometimes - data composition is a very hard problem.
Data Pre-processing	Ease	Fix missing data and outliers - very easy!	Hard. Requires deduplication toxicity filtering diversity management, quantity control
Model Learning	Model Size	ML models, such as classifiers or regressors, typically have at max around 200 million parameters and are generally less computationally intensive for experimentation than LLMs.	LLMs typically have 100 billion or even trillions of parameters, making them significantly larger and more complex than many nongenerative models. Their massive scale impacts resource requirements, storage, and

Lifecycle Step	Context	MLOps	LLMOps
			computational efficiency.
Hyperparameters	Accuracy	Limited hyperparameter space makes search easy making them ideal for predictive problems	Massive hyperparameter space leads to real-time search latency. This makes them ideal for generative and evolutionary tasks that require creativity. Accuracy can be a computational bottleneck.
Model Training Duration	Training Scale and Duration	Less resource-intensive and generally faster. Can be easily deployed as notebooks on the cloud or as containerized solutions across a single node.	Involves processing massive datasets with distributed training across large clusters of GPUs or TPUs and optimizing for parallel processing. Can take days or weeks. Operationalizing LLMs may require dynamic

Lifecycle Step	Context	MLOps	LLMOps
			resource scaling and involves designing scalable infrastructure and orchestration systems to handle varying workloads efficiently.
Domain Adaptation	Cost	Full fine-tuning is essential and affordable	Full fine-tuning is too expensive and pretty rare. Instead, popular techniques are:  Prompt engineering RAGs Knowledge graphs Parameter-efficient fine-tuning
Evaluation	Ease	Easy Discriminative Well-defined probability space	Extremely Hard Problem  Generative and thus hard to detect

Lifecycle Step	Context	MLOps	LLMOps
			Unbounded probability space
Robustness	Static versus Dynamic	Model behavior stays the same in production	Model behavior changes in production based on interactions and requires constant monitoring for alignment.
Security	Secure	Highly secure	<ul style="list-style-type: none"> <li>- Highly vulnerable</li> <li>- Needs a DataSecOps framework</li> </ul>

## Four Goals for LLMOps

LLMOps operates on an LLM-specific set of design-pattern principles to ensure that your LLM applications achieve four key goals: scalability, safety, reliability, and robustness. Let's take a closer look at these goals:

### *Safety*

Minimizing the regulatory, reputational, and operational risks associated with deploying LLM applications in production

### *Scalability*

Building, storing, and maintaining LLM applications that can generalize across data and scale efficiently on demand while optimizing latency,

throughput, and costs

### *Robustness*

Maintaining high performance of LLM applications over time in the face of data drift, model drift, third-party updates, and other challenges

### *Reliability*

Implementing rigorous inference monitoring, error handling, and redundancy mechanisms to prevent downtime and failures

LLMOps teams automate repetitive processes to more quickly optimize these applications at scale and avoid those “LLM-oops!” moments. Another key aspect of the LLMOps framework is fostering consistency, transparency, and collaboration between diverse interdisciplinary teams. These teams often include data engineers, data scientists, research scientists, software engineers, and business operations teams.

LLMOps is an entirely new field, so, as of this writing in 2024, there are very few mature tools and resources available. The LLMOps teams at various organizations are thus internally developing their tools and processes, based on prototypical open source libraries and toolkits.

## **LLMOps teams and roles**

Today, there are two kinds of companies out there building with LLMs: the newer startups, which are primarily focused on LLM applications, and companies with existing ML models that are now building their own GenAI teams.

In the latter category, there are so few skilled Ops professionals that most companies recruit ML engineer candidates internally and then upskill them to LLMOps, instead of hiring externally. A major cause is the general lack of clarity around use cases as well as expected job responsibilities. So the current norm is hiring 8 to 10 people internally from different departments, which could include product managers, full-stack engineers, system

architects, data engineers, data scientists, ML engineers, platform engineers, cybersecurity professionals, and developer advocates. Many companies want someone who already understands the business inside and out to test the feasibility of several potential use cases and projects before committing to one.



IMAGE TO COME

Newer startups, however, have no choice but to build a new team from the ground up. These teams can look very different, depending on if they are working on LLMOps infrastructure (LLMOps SaaS companies) or LLM use cases such as copywriting, education, or process optimization. LLMOps teams come in all shapes and sizes, with different levels of business and technical maturity. (Later in the chapter, we'll look at how to assess your company's LLMOps maturity.)

Most companies cannot afford the cost of building and training an LLM for use in their application. Instead, most choose to use a foundational model. This is where AI engineers come in: to build a quick proof of concept using LLMOps tools. Most come from a software-engineering background: they know how to build a full-stack application quickly, but don't necessarily have a deep understanding of ML/LLM models' inner workings or how to optimize them.

Once the proof of concept is completed, building, deploying, and optimizing ML models for the business falls to LLM engineers, ML scientists (interchangeable names, depending on the organization), and LLMOps Engineers tasked as reliability engineers.

If a team chooses to deploy an LLM application using API integration, the initial deployment will be pretty straightforward. The LLMOps engineers work alongside AI engineers to scale the model and improve its performance. Ideally, each AI engineer should be paired with an LLMOps engineer; the AI engineer can focus on adapting and fine-tuning the model to meet the business needs, while the LLMOps engineer manages the deployment and optimization. For open-source models, managing deployment automatically falls to the LLMOps engineer.

As the tech industry moves from nongenerative models to generative models, it is shifting away from *feature engineering*, or creating features to model the data and experimenting with different hyperparameters to optimize performance. Generative models, and specifically LLMs, do not require feature engineering. Today, the core requirements are usually prompt engineering or building a RAG pipeline: skills that lie within the domain of AI engineers.

Another big shift is that the data-engineering pipeline and the monitoring and evaluation pipeline have become far more complex. Evaluating LLMs is much more than straightforward quantitative scoring. Some industry benchmarks exist—like BLEU (Papineni et al. 2002), a benchmark used to evaluate machine translation, and ROUGE (Lin 2004), a benchmark used to evaluate summarization—but they are loosely correlated with application performance. Moving to a model that has a better score is no guarantee of better user satisfaction. Standardized scores may be good for comparing LLMs in general, but ultimately users care about whether the LLM application is solving their problem.

In addition, since LLMs are deployed in consumer-facing applications, metrics like perceived latency and throughput become make-or-break factors in market competition. The model is not the only deciding factor: the deployment, evaluation, and monitoring pipelines are also at the front and center of performance assessment.

Let's look at some of the roles involved in these pipelines:

*Data engineer*

Data engineers are professionals responsible for designing, building, and maintaining systems and pipelines that enable the efficient collection, storage, transformation, and access to data. Their work ensures that data is available, reliable, and organized in a way that allows data scientists and other engineers to create and evaluate models and data-driven applications. Data retrieval and movement are the most fundamental skills for data engineers, but as they progress in their career, developing data architectures becomes more important.

Data engineering for LLMs requires specialized understanding: how to chunk the data, what tokenization model to use, and so on. Thus, it's best to pair each data engineer with an ML scientist (with an LLM engineer background), along with the LLM Ops engineer for automating and streamlining LLM systems at scale.

### *AI engineer*

The core skillset of an AI engineer is full-stack engineering and development (React, Node.js, Django), plus the common LLM Ops tools and frameworks for deploying applications, like LangChain and Llama Index. They need a foundational understanding of end-to-end AI application development, including prompt engineering and RAG systems. Companies building their teams from scratch usually look to hire AI engineers who also know a lot about using cloud services to deploy and manage AI applications and have experience working with external APIs and vector databases.

### *ML scientist*

The day-to-day work of an ML scientist, NLP scientist, or LLM engineer involves researching, designing, and optimizing LLMs using frameworks like PyTorch, TensorFlow, and JAX. This role requires a deep understanding of NLP algorithms and tasks, such as tokenization, named entity recognition (NER), sentiment analysis, and machine translation. Candidates should know model architectures and training and fine-tuning processes.

## *LLMOps engineer*

The goal of an LLMOps engineer is to ensure that LLM applications remain reliable, robust, secure, and scalable. The day-to-day work involves building and maintaining operational LLM pipelines as a project owner.

The next section looks at the LLMOps engineer role in more detail.

## **The LLMOps engineer role**

This role requires extensive expertise in deploying, monitoring, fine-tuning, training, scaling, and optimizing LLM models in production environments; infrastructure and platform engineering; data engineering; and system reliability.

Companies building LLM teams typically seek LLMOps engineers who understand the unique challenges associated with LLMs and are experienced in making build vs. buy tradeoffs, weighing cost efficiency against system performance.

To stand out as a candidate for this role, you'll need proficiency in a unique blend of specialized skills and a deep technical understanding of the entire LLM lifecycle, from data management to model deployment and monitoring. You'll also need to be a problem-solver, a strong team player and communicator, and meticulously detail-oriented.

To illustrate what this means in practice, let's look at a typical day in the work life of a fictional LLMOps engineer.

## **A Day in the Life**

While the specific responsibilities for this role will vary across organizations, the core tasks typically encompass a blend of infrastructure management, collaboration, optimization, and compliance. To give you a quick taste of what this looks like in practice, here's what a typical LLM engineer's workday might look like:

7:30 AM – 8:30 AM: Morning check-in

- Review monitoring dashboards for any overnight alerts or performance issues in deployed LLMs; address any urgent issues or escalate them to the appropriate teams.
- Lead the team's daily stand-up meeting to discuss ongoing projects, blockers, and priorities for the day.

8:30 AM – 10:00 AM: Infrastructure management and optimization

- Modularize code for reusability , creating separate modules for provisioning GPUs, managing storage, and networking.
- Implement batching mechanisms to process multiple inference requests, reducing the per-request overhead
- Implement latency optimization techniques like kernel fusion, quantization, and dynamic batching to enhance model performance.

10:00 AM – 11:30 AM: Collaboration and project planning meeting

- Meet with data scientists, ML engineers, and Red Teaming engineers to discuss usage requirements, timelines, monitoring errors, and scaling challenges.

11:30 AM – 12:30 PM: API development and model deployment

- Designing inference end-points and cache for different models in production ensuring compatibility with the rest of the application

12:30 PM – 1:30 PM: Lunch break

1:30 PM – 3:00 PM: Monitoring and troubleshooting

- Troubleshoot any issues that arise, for example, let's say the users are experiencing long delays when making requests to the inference API, they would identify the source of latency by examining hardware utilization, and network latency, examine

usage logs, and then implement a solution that could be using a Pod Autoscaler or caching the frequent requests etc.

3:00 PM – 4:30 PM: Research and continuous learning

- Experiment with new tools, libraries, or frameworks that could be integrated into the existing tech stack to improve efficiency and performance.

4:30 PM – 5:30 PM: End-of-day wrap-up, review, and on-call prep

- Review the day's tasks and update the tickets with completed tasks and next steps.
- Prepare for any on-call duties, ensuring that all monitoring systems are correctly configured and that you're ready to respond to any incidents.
- Attend any final meetings or syncs with cross-functional teams to ensure alignment on upcoming priorities.
- Wrap up any remaining tasks and make sure that all systems are running smoothly before logging off for the day.

After regular working hours, if you are on call, you'll need to remain available to address any critical issues that may arise.

## **Hiring an LLMOps engineer externally**

There are two ways to fill an LLMOps engineer role: hiring externally, or hiring internally and upskilling your people, training ML engineers to become LLMOps engineers. This section will look at external hiring first, and then discuss upskilling existing employees.

If you're hiring for this role, other skills to look for in candidates for LLMOps engineering roles include experience or proficiency in:

- Converting models to and from libraries like PyTorch or JAX

- Understanding ML metrics like accuracy, precision, recall, and PR-AUC
- Understanding data drift and concept drift
- Running and benchmarking models to understand the impact of computational graph representation on performance across the neural engine, GPU, and CPU
- Deploying and scaling ML models in cloud environments like AWS, GCP, and Azure
- LLM inference latency optimization techniques, including kernel fusion, quantization, and dynamic batching
- Building Ops pipelines for data engineering, deployment, and infrastructure as code (IaC) using tools like Terraform, managing vector databases, and ETL processes for large-scale training datasets
- Understanding red-teaming strategies, interfaces, and guidelines
- Using Docker for containerization and Kubernetes for orchestration to ensure scalable and consistent deployments
- Collaborating and managing projects with teams that include LLM engineers, data scientists, and ML/NLP engineers



IMAGE TO COME

Every company, of course, has its own interviewing process. Some conduct many rounds of interviews; others combine several of the interviews

described below into a single on-site meeting. This section describes a fairly standard four-round interview process.

## **Round 1: Initial Screening**

The goal during initial screening is to determine that each applicant has the fundamental skills and experience required for the role. This can be assessed via a resume assessment sheet. The questions you're asking here are very high-level: Do they have experience with deploying LLMs in production environments? Do they mention specific frameworks and tools for managing LLM pipelines, and are these the same tools your company uses? If not, will they be able to learn and adopt your stack? Can they show or talk about any past projects?

## **Round 2. Technical Assessment**

The goal in this round is to assess the candidate's technical proficiency in core areas such as LLM deployment, data engineering, and infrastructure management. Some questions you might ask include:

- Describe the steps you take to fine-tune a pre-trained large language model. How do you ensure that the model is optimized for the specific use case?
- Walk me through the deployment process of an LLM you've worked on. What challenges did you face? How did you overcome them?
- How would you set up a CI/CD pipeline for LLM training, fine-tuning and deployment?
- How do you design and manage data pipelines for large-scale ML projects? What tools do you use, and how do you ensure data quality?
- How do you monitor and troubleshoot latency issues in production?

- How do you handle versioning and tracking datasets used in training or fine-tuning LLMs?
- How do you ensure high availability and cost-efficiency in your cloud infrastructure?

### **Round 3. System Design Interview**

The goal in this round is to assess the candidate's ability to design scalable, reliable, and maintainable systems for deploying and managing LLMs.

Sample questions for this round could include:

- Tell me about a time when you had to make a build vs. buy decision for a component of your ML infrastructure. What factors did you consider?
- How would you design an API for serving LLM inferences at scale? Discuss considerations for load balancing, fault tolerance, and latency reduction.
- How would you use an IaC tool to manage the cloud infrastructure for an LLM deployment? What strategies would you employ to optimize resource usage and cost? What potential problems do you think you'd encounter while scaling it?
- Describe your approach to dynamic batching in inference service. How do techniques like quantizing and mixed precision training affect the performance and efficiency of LLMs?
- How do you manage memory in CUDA when training LLMs? What strategies do you use to prevent issues like out-of-memory errors?
- How do you benchmark model performance before and after optimization? What metrics do you consider, and what tools do you use?
- How would you diagnose and address performance degradation after an optimization?

## **Final Round: Behavioral Interview**

In the fourth round, you've narrowed the pool to the most qualified candidates. Now you need to assess their personalities: Are they self-driven? Can they work in a team, handle challenges, and contribute to a collaborative environment? Sample questions:

- How do you stay up to date on the latest advancements in LLMs and machine learning operations?
- How do you approach integrating data scientists' feedback into the deployment process?
- How would you collaborate with a red-teaming engineer to address potential security vulnerabilities in an LLM deployment?
- What experience do you have with on-call rotations? How do you handle critical incidents during off-hours?

You'll also want to ensure the candidate aligns with your organization's values, particularly in areas like innovation, continuous learning, and collaboration. You can ask questions like:

- What are your favorite newsletters in the space?
- How do you keep up with new advances in the field?

## **Hiring internally: Upskilling an MLOps engineer into an LLMOps engineer**

The gap between MLOps engineers and LLMOps engineers is significant in terms of scale, complexity, and the technical challenges involved. Thus, upskilling an existing employee requires a focused effort to build their understanding.

That said, the foundational skills of MLOps—such as model deployment, automation, and cloud management—provide a solid base from which to grow. With dedicated learning and hands-on experience, an MLOps engineer can transition into the LLMOps domain effectively. The core

upside of hiring internally is that candidates are already aligned with the organization's values and culture and have a keen understanding of its KPIs.

To excel, new LLMOps engineers need resources and training to deepen their understanding of large-scale model architectures and transformer architectures, attention mechanisms, infrastructure management, and LLM-specific optimization techniques. They also need to understand LLMs differ from non-generative ML models. We recommend pairing them up with LLM engineers to experiment with and evaluate different models.

This role isn't just about building an app that uses LLMs. LLMOps engineers also manage the balance between cost, cloud resources, and user experience and handle huge unstructured datasets. Pair them with data engineers to help build a data processing pipeline, so they can familiarize themselves with the data sources at their disposal, how the data is structured in the databases, how different databases retrieve information, what the company's latency expectations are, and how to handle data filtering. Allow them to introduce multi-node setups and distributed systems for different models while focusing on cost optimization and errors. Get them to benchmark different LLM models and debug their performance optimization. Finally, allowing them to present their logging practices and the guardrails they have set up for maintaining reliability and performance at scale.

Most MLOps engineers already have skills in model versioning, data versioning, managing rollbacks, and GitHub Actions, so upskilling them can be an effective strategy for building a strong LLMOps team.

Next, let's look at how to make sure your LLMOps engineers' goals are aligned with your organizational goals.

## **LLMs and Your Organization**

You learned at the beginning of this chapter that the four key goals of the LLMOps framework are safety, scalability, reliability, and robustness. For LLMOps teams, then, the next big question is how to measure the

application's performance against those goals. How will you know you're succeeding? The company's expectations should be clearly defined and remain quantitatively as well as qualitatively measurable at all times.

Three kinds of metrics will allow you to measure your team's performance toward its goals: SLOs, SLAs, and KPIs. These terms are in common usage among site reliability engineers, but LLMOps teams are rapidly adopting them as well.

### *Service level objectives (SLOs)*

Service level objectives are specific, measurable targets set by an organization to gauge the quality of its services internally. They define what level of service the organization aims to achieve. For example, an SLO for a cloud hosting company might be to ensure that server uptime is at least 99.9% per month.

### *Service level agreements (SLAs)*

Service level agreements are formal contracts between a service provider and a customer that define the level of service the provider commits to deliver. They typically include specific performance goals and stipulate remedies if those metrics are not met. For example, if the uptime for an internet provider falls below 99.9% annually, the SLA might stipulate that the customer will receive a 10% discount on their next billing cycle.

### *Key performance indicators (KPIs)*

Key performance indicators measure the overall success and performance of specific business activities. They provide insight into how well the organization is achieving its strategic objectives. For example, an important KPI for an app might be its churn rate or the percentage of customers who stop using the app over a certain period.

In August 2024, a **Gartner Research study predicted** that 30% of currently existing GenAI projects would fail by 2025. (In fact, Gartner **published**

similar findings in 2018, predicting that 85% of ML projects would fail in production by 2022.) The key failure points outlined in the 2024 study are notable because they are the operational aspects of LLM development and deployment—including data-quality issues, the lack of a strong evaluation framework, and the high costs of scaling these models in production.

That said, one of the most obvious issues is mismatched expectations between management and engineering teams. For the last ten years, one of data scientists' biggest skill gaps has been translating ML model metrics into organizational and product success metrics. In other words, when you're measuring abstract goals like model security, scalability, robustness and reliability, how do you communicate what this means for the business? That's what SLOs, SLAs, and KPIs are for.

Using a SLO-SLA-KPI framework allows LLMOps teams to automate, streamline and manage expectations across multiple stakeholders. SLOs make it evident to all stakeholders what level of service is being aimed for. SLAs ensure accountability so that everyone involved is aware of their roles and the agreed-upon service levels. This can also help you track performance and address any deviations from the expected standards. And KPIs provide visibility into real-time data to help detect potential issues early, facilitating informed decision-making.

Let's look more closely at the LLMOps goals to see how these metrics translate.

## **Reliability**

As you know well by now, LLMs are extremely complex, with billions of parameters. Their behavior can be unpredictable, and they sometimes exhibit unexpected responses or errors due to their scale and the intricacies of their training data. Additionally, if the training data is biased, outdated, or unrepresentative of certain domains, the model's performance can be unreliable in those areas.

LLMs also struggle at times with context, nuance, and understanding the intent behind user queries. This can lead to incorrect, irrelevant, or

misleading responses. What’s more, LLMs usually aren’t updated in real time. As language evolves, if new information is not integrated into the training data via regular retraining, models can become outdated, leading to decreased reliability.

All of these issues come down to reliability. LLM-based applications’ reliability can be measured in terms of system availability, error rates, and customer satisfaction. Table 2-2 shows how these metrics would look as SLOs, SLAs, and KPIs.

Table 2-2. Example: Reliability SLOs, SLAs, and KPIs.

	SLO	SLA	KPI
Availability	Maintain 99.9% uptime per month	Ensure that the system is available for at least 99.95% of requests over a rolling 30-day period	Customer Satisfaction Score (CSAT) related to system availability
Error Rate	Keep the error rate below 0.1% for all API requests	Ensure that less than 1% of user interactions result in errors	Track error-rate trends over time and analyze root causes of major errors
Customer Satisfaction	CSAT score of at least 90%.	Ensure that the Net Promoter Score (NPS) remains above 8	Post-interaction surveys or feedback forms; CSAT score

## Scalability

LLMs tend to have a large memory footprint, often exceeding the capacity of a single machine. Distributing a model across multiple GPUs or nodes while maintaining performance is technically challenging. Handling large volumes of data efficiently is critical.

Another significant challenge is scaling the data pipeline to feed data into the model at the required speed without causing bottlenecks. This can be especially hard for applications like chatbots or interactive services, where low latency is crucial. Scaling while keeping response times low can be challenging, since scaling up increases resource contention and network overhead. Therefore, balancing performance with cost efficiency is a constant concern.

LLMOps scalability can be measured via metrics like latency, throughput, response time, resource scaling, capacity planning, and recovery time objective (RTO), as shown in Table 2-3.

Table 2-3. Example: Scalability SLOs, SLAs, and KPIs.

	SLO	SLA	KPI
Latency	95% of requests should be served within 200 milliseconds.	Keep the average response time for API calls under 100 milliseconds	Average response time for user interactions
Throughput	Process a minimum of 1,000 requests per second during peak traffic hours	Handle at least 1 million concurrent connections without degradation	Peak throughput capacity during load testing
Response Time	Maintain a web page load time of under 3 seconds for 95% of users	Ensure that the login process completes within 500 milliseconds for 99% of users	User-experience metrics related to response times
Resource Scaling	Automatically scale up resources to handle a 50% increase in traffic within 5 minutes	Ensure that adding additional servers linearly increases throughput without impacting latency	Scalability test results and cost-effectiveness of scaling solutions
Capacity Planning	Maintain CPU utilization below	Ensure that there are enough database	Resource utilization trends and

	SLO	SLA	KPI
	80% during peak hours	connections available to handle double the anticipated peak load	forecasting accuracy
Recovery Time Objective (RTO)	Achieve a recovery time objective of under 30 minutes for critical system failures	Ensure that the system can recover from a database failure and restore service within 15 minutes	Historical RTO metrics and improvement initiatives

## Robustness

Over time, the statistical properties of the model’s training data can change, leading to a drift in the model’s performance. This is particularly problematic for models that interact with real-time or rapidly changing data. This can lead to performance degradation in the form of outdated or irrelevant responses.

Continuous training and fine-tuning are necessary to maintain robustness, but they require significant computational resources and careful management to avoid introducing new biases or errors. You can measure it via metrics like data freshness, model evaluation, and consistency, as shown in Table 2-4.

Table 2-4. Example robustness SLOs, SLAs, and KPIs

	SLO	SLA	KPI
Data Freshness	Ensure that dashboard data is refreshed every 5 minutes	Guarantee that data is updated in real-time	Data refresh latency and accuracy of real-time data updates
Model Evaluation	Maintain a performance degradation rate of less than 5% over 6 months	Guarantee regular updates and reviews of model evaluation metrics	Accuracy, relevance, and update frequency of evaluation metrics
Consistency	Guarantee strong consistency for data reads and writes across all regions	Maintain eventual consistency with a maximum propagation delay of 1 second	Consistency model adherence and replication latency

## Security

Maintaining LLM application security is challenging: these are complex models handling sensitive data in the face of constantly evolving security threats. LLMs are especially vulnerable to adversarial attacks, data poisoning, and other forms of exploitation that can compromise their integrity and security.

Managing and controlling access to the LLM and its data is complicated, especially in a multi-access or multi-tenant environment, but it's critical for preventing unauthorized access and misuse. Table 2-5 shows some ways to measure it.

*Table 2-5. Example security SLOs, SLAs, and KPIs*

	<b>SLO</b>	<b>SLA</b>	<b>KPI</b>
Data Privacy	Ensure data encryption for all in-transit and at-rest data.	Ensure no breaches occur	Encryption compliance status.
Model Integrity	Detect and address any model tampering within 24 hours.	Guarantee prompt detection and response to unauthorized modifications.	Number of unauthorized modifications detected.
Access Control	Achieve a user authentication success rate of 99.9%.	Ensure robust user authentication and authorization mechanisms.	Rate of unauthorized access attempts.
Red Teaming	Ensure detection of 99.9% of attempted adversarial attacks.	Ensure regular security assessments and updates.	Frequency of security assessments and the number of critical vulnerabilities identified.

When all teams—whether they are involved in development, operations, or management—understand the agreed-upon service levels and performance indicators, they can work together more effectively toward common goals.

This alignment fosters a unified approach to managing and improving project performance. It also helps in making data-backed decisions about resource allocation, process changes, and strategic adjustments. Most importantly, it helps in building trust and ensuring that everyone is on the same page regarding expectations and outcomes.

Overall, implementing an SLO-SLA-KPI framework not only enhances transparency and fosters collaboration—it also serves as a foundational element in evaluating and advancing the maturity of your LLMOps practices, which is the topic of this chapter’s final section.

## The LLMOps Maturity Model

*LLMOps maturity* is a way of determining how well an organization’s LLM operations align with industry best practices and standards. Assessing LLMOps maturity helps organizations identify their strengths, areas for improvement, and opportunities for scaling and enhancing the robustness of their LLM systems.

A couple of years ago, Microsoft published a “**machine learning operations maturity model**” detailing a progressive set of requirements and stages to measure the maturity of MLOps production environments and processes. The LLMOps Maturity Model we present here, inspired by Microsoft’s MLOps model, is meant to do the same for LLMOps teams. Although this is by no means a comprehensive audit, we hope to see several variations put into practice.

The three LLMOps maturity levels are as follows:

### *Level 0*

No LLMOps practices are implemented. The organization lacks formal structures and processes for managing and deploying its LLM systems, which hinders its effectiveness.

### *Level 1*

The organization applies MLOps practices but without LLM-specific adaptations. This is an improvement over Level 0 in terms of formalization and processes but still lacks the sophistication needed for full LLM operations.

### *Level 2*

Achieving Level 2 represents a mature LLMOps state, characterized by advanced documentation, robust monitoring and compliance measures, and the integration of sophisticated orchestration and human review strategies. Usually, this can be assessed by asking some questions about whether decision strategies and model performance measures and metrics are well documented within the team.

*Table 2-6. Documentation and strategy measures of LLMOps maturity*

	<b>Level 0: No LLMOps</b>	<b>Level 1: MLOps, No LLMOps</b>	<b>Level 2: Full LLMOps</b>
Are the business goals and KPIs of the LLM project documented and kept up to date?	Not documented	There is documentation, but it's often outdated	Full documentation with regular updates; KPIs include model performance metrics, operational efficiency, and cost-effectiveness
Are LLM model risk-evaluation metrics documented?	No formal risk assessment	Basic risk evaluation for model accuracy and data security	Comprehensive risk evaluation including bias, fairness, data drift, and performance degradation, with mitigation strategies in place
Is there a documented and regularly updated overview of all team members	No documentation	High-level roles are documented, but responsibilities	Detailed team structure with roles, responsibilities, and contact

	<b>Level 0: No LLMOps</b>	<b>Level 1: MLOps, No LLMOps</b>	<b>Level 2: Full LLMOps</b>
involved in the project, along with their responsibilities?		may be unclear for the newer roles	details, regularly updated and reviewed
Is the choice of LLM well-documented and cost-compared against other open-source/proprietary offerings?	No documentation or cost analysis	Basic documentation of LLM choice, minimal cost comparison	Detailed documentation including rationale for choice, performance benchmarks, and cost comparison against alternative models
Is the API for the model vendor well-documented, including request and response structure, data types, and other relevant details?	No API documentation	Model developed in-house	Comprehensive API documentation, including request/response examples, data types, error codes, and versioning details
Is the software architecture well	No documentation	There is a high-level architecture	Detailed architecture diagrams,

	<b>Level 0: No LLMOps</b>	<b>Level 1: MLOps, No LLMOps</b>	<b>Level 2: Full LLMOps</b>
	documented and kept up to date?	overview, but it may be outdated	including data flow, system components, and integration points, updated regularly

Documenting the above factors can be incredibly helpful when choosing and deploying any new model. For example, given the significant costs that are associated with deploying an LLM application, cost-benchmark analysis documentation allows the company to decide which model to roll into production and estimate the project timeline.

After deployment, the company also needs to assess how well the team has documented the model’s performance measures and metrics. This is to make sure that everyone on the team understands the expectations and that they are comprehensively monitoring the model performance in production.

Table 2-7 outlines three levels of LLMOps maturity with regard to model performance and evaluation. Keeping these different levels in mind, organizations and the LLMOps team will be better prepared to deal with contingencies and can better align projects with business goals, mitigate risks, and enhance operational efficiency.

Table 2-7. Model performance and evaluation measures of LLMOps maturity

	Level 0: No LLMOps	Level 1: MLOps, No LLMOps	Level 2: Full LLMOps
Does the LLM system operate within its knowledge limits, and recognize when operating outside those limits?	No mechanisms to detect limits	Basic detection of operational limits	Advanced guardrails, limit-detection mechanisms, and documentation of context-aware warnings using techniques like confidence scoring and thresholding
Are the LLM’s inputs and outputs automatically stored?	No automatic storage	Basic storage of inputs and outputs	Automated storage of all inputs and outputs with indexing for easy retrieval and analysis
Is A/B testing performed regularly?	No A/B testing	Occasional A/B testing with limited coverage	Regular A/B testing with comprehensive test coverage and analysis, using tools like Optimizely or

	<b>Level 0: No LLMOps</b>	<b>Level 1: MLOps, No LLMOps</b>	<b>Level 2: Full LLMOps</b>
			custom frameworks
Are all API requests and responses logged, and API response time and health status monitored?	No logging or monitoring	Basic logging and response time monitoring	Comprehensive logging with detailed request/response analysis; real-time health monitoring using tools like ELK stack
Is the LLM monitored for toxicity and bias?	No outlier detection or bias monitoring	Basic outlier detection with manual review	Advanced automated toxicity and bias detection pipelines using statistical methods and regular bias audits, with automated alerting for low-confidence predictions
Are processes in place to ensure that LLM operations	No process exists	Existing Process to ensure that LLM	Existing process to ensure that LLM operations comply with

	<b>Level 0: No LLMOps</b>	<b>Level 1: MLOps, No LLMOps</b>	<b>Level 2: Full LLMOps</b>
	comply with regulations such as GDPR, HIPAA, and other relevant data protection laws?	operations comply with regulations such as GDPR, HIPAA, or other relevant data protection laws	regulations such as GDPR, HIPAA, or other relevant data collection and protection laws and copyrighting laws
Does the LLM-based app use anonymization to protect users' identities while maintaining the data's utility for LLMs?	No anonymization	Basic anonymization techniques applied	Advanced automated anonymization methods, including data masking and aggregations
Does the organization perform regular security reviews and audits of LLM infrastructure and code?	No regular reviews	Periodic security reviews and audits	Regular, comprehensive security reviews and audits, including third-party assessments and vulnerability scans

At Level 0: No LLMOps, machine learning efforts are often isolated and experimental, and lack any systematic deployment and monitoring framework. The models may be developed in silos, often resulting in

unreliability and inefficiency. Chevrolet's chatbot blunder is an excellent example of this that due to the lack of monitoring and guardrails was abused by the community for algebra homework, offering Chevrolet cars in no take-backsies deals and promoting Tesla cars instead.

At Level 1: MLOps, no LLMOps, the organization is likely to have a robust pipeline for model training, testing, and deployment, with automated monitoring and retraining workflows. However, these are builders for smaller models and not fully optimized for the specific challenges of LLMs.

At Level 2: Full LLMOps, the highest level of maturity, the organization has adopted MLOps practices and is fully optimized for LLM applications. Its infrastructure is capable of handling large-scale LLM deployments, fine-tuning, real-time inference, auto-scaling, and resource management. Mature LLMOps teams have failover and rollback mechanisms in place and can act quickly if the updated model underperforms after deployment. The organizations can deliver more reliable responses, be efficient with ROI, and reduce operational risks.

## Conclusion

In this chapter, we discussed the team structure for organizations building LLM applications. We discussed various roles and how to build a highly effective team. Finally, we discussed a framework for typing the LLM performance metrics with the business KPIs. In the next chapter, we will talk about how LLMs have changed the data engineering landscape, and we'll show you how to build performant data pipelines for LLMs.

# Chapter 3. The Future of LLMs and LLMOps

---

## A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 10th chapter of the final book. The GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at [sgrey@oreilly.com](mailto:sgrey@oreilly.com).

In the next decade, the future of LLMOps, LLMs, NLP, and knowledge graphs will converge in ways we can barely imagine today. Imagine this - AI systems no longer as distant tools but deeply integrated into every facet of our lives. Even the most popular LLMs today are somewhat clunky iterations (Tu et al. 2024), but in the near future, we believe they will be refined to a point where their understanding of language will rival human intuition (Amodei 2024; Hagendorff, Fabi, and Kosinski 2023a). This is because of emergent traits in LLMs (Hagendorff, Fabi, and Kosinski 2023b).

Currently, the main way users interact with LLMs is through text-based chats, but in the upcoming years, they won’t just be answering questions; they’ll be engaging in complex problem-solving, offering insights, and pushing the boundaries of creativity itself. For example, on September 25, 2024, OpenAI released the Advanced Voice Mode for its ChatGPT



this area. LLMOps engineers will spend less time debugging code and more time refining high-level system strategies, including platform and infrastructure designs that automatically balance model training with compute costs.

These models will adapt on the fly, learning from real-world feedback at a rate that feels almost magical. This has been the primary goal of AutoML, which is an active area of research in ML. Most importantly, as LLMs start generating higher-quality translated content, they can more easily expand their capabilities to additional languages, even less common ones. In addition, LLMs are getting better at using multimodal inputs. Currently, it takes an LLM a few seconds to process text, voice, and images, and a lot of progress is being made in simultaneous speech-to-text translation (Papi et al. 2024), a critical step in speech-to-speech translation. Once simultaneous speech-to-text quality is high enough, existing text-to-speech models such as OpenAI Whisper can be used to complete the speech-to-speech translation pipeline.

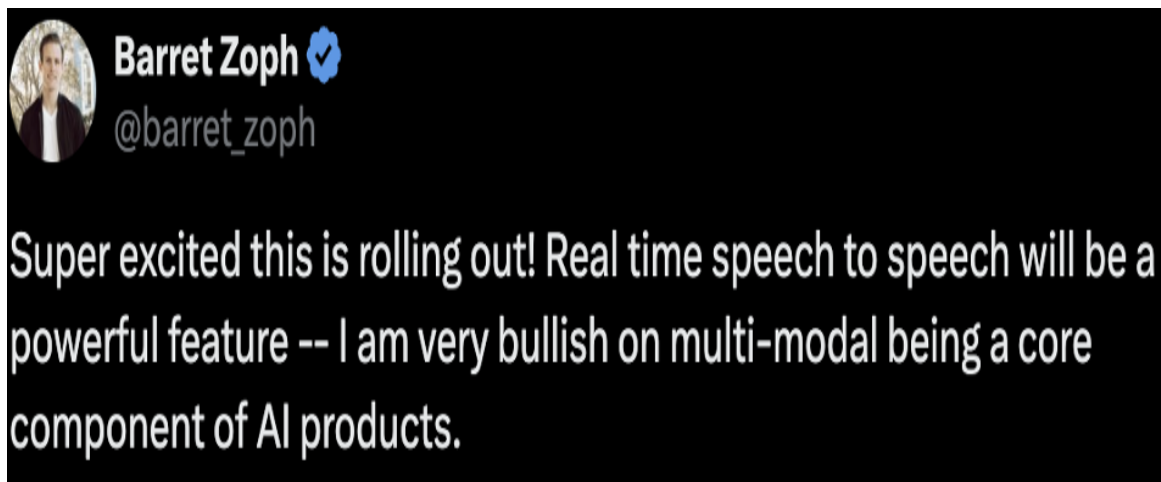


Figure 3-2. *X post* by Barret Zoph on September 24, 2024, during his time as Research VP at OpenAI

One of the limitations of LLMs is that, at their core, they generate the most likely next word based on the data they were trained on and the prompt submitted, but they don't seem to understand even simple concepts. For a popular example (pictured in Figure 10-3), Meta's LLM could easily tell that Tom Cruise's mom is Mary Lee Pfeiffer, but had trouble with the question "Who is Mary Lee Pfeiffer's famous actor son?" It frequently

answered with the names of other famous actors, such as Matt Damon, Tom Hanks, and Michelle Pfeiffer—the “famous actor” part of the prompt took precedence.



Who is Mary Lee Pfeiffer's famous actor son?



Mary Lee Pfeiffer is the mother of actor Derek Pfeiffer, but I couldn't find any information about a famous actor son of hers. However, I did find that Mary Lee Pfeiffer is the mother of Michelle Pfeiffer, a renowned American actress known for her iconic roles in films like "Scarface," "The Fabulous Baker Boys," and "Batman Returns."

Who is Tom Cruise's mom?



Tom Cruise's mother is Mary Lee Pfeiffer.



Ask Meta AI anything...



Messages are generated by AI and may be inaccurate or inappropriate. [Learn more](#)

*Figure 3-3. Meta's AI answers to questions about actor Tom Cruise and his mother, Mary Lee Pfeiffer, show a lack of conceptual understanding.*

One solution that is under research is using **knowledge graphs** that contain relationships between concepts. Knowledge graphs contain representations of interconnected concepts and their relationships. For example, Wikipedia lists Mary Lee Pfeiffer as Tom Cruise's mother, but the fact that he is her son is not written down: it's implicit. Knowledge graphs make relationships explicit, creating systems that understand context in a way that's almost human (Chen et al. 2024; Pan et al. 2024)

Conversations with LLMs will become indistinguishable from human conversations. No more fumbling with chatbots or dealing with "robotic" responses. Advances in personalization can allow future LLM applications to combine several facts they learn about users in the course of their interactions. Simple versions of this already exist today. For example, ChatGPT already learns what programming language an individual user frequently asks questions about and provides answers in that language by default. Recent research (Zhang et al. 2024) provides several other examples of adding personalization for education, healthcare and finance. Although they are currently mostly limited to research papers, when deployed successfully in production across every e-commerce application out there, LLMs will anticipate users' needs, contextualise interactions, and even predict trends, all while learning continuously from new data streams.

One of the key concerns people voice about LLMs has to do with the alignment risks associated with these models: If a model is capable of showing emergence and can predict and emulate human behaviour, then how do we know it will continue to be helpful in the long run? Should development pause while researchers closely monitor these models' performance? How can we ensure that the model is not behaving in a sociopathic way, providing harmless answers only when it realizes it's being tested? Researchers are exploring answers to these questions (Ji et al. 2024).

# The Future of Large Language Models

LLM architecture has advanced exponentially from the mid-2010s to the mid-2020s, but the coming years promise even more profound shifts. These changes will redefine not only how LLMs are structured but also how they interact with data, humans, and each other [9]. From increased scalability to more efficient computation, from hybrid architectures combining multiple paradigms to emergent self-learning systems, the future of LLMs has breathtaking potential. This chapter explores several aspects of that potential.

## Scaling Beyond Current Boundaries

Today's LLMs, like GPT-4, have reached impressive scales, but they are far from their limits. Going into the 2030s, we expect to see architectures designed with scalability in mind from the ground up. This is not simply about increasing the number of parameters; it's about efficient, targeted scaling. Future architectures will incorporate hierarchical layers of models (also known as *Hierarchical Attention Networks*), where each layer is optimized for a specific domain of understanding, such as reasoning, emotion, or even creativity.

## Training at scale:

- GPU falling off the bus
- GPU Driver stuck
- DRAM UCE
- SRAM UCE
- Network cable failure

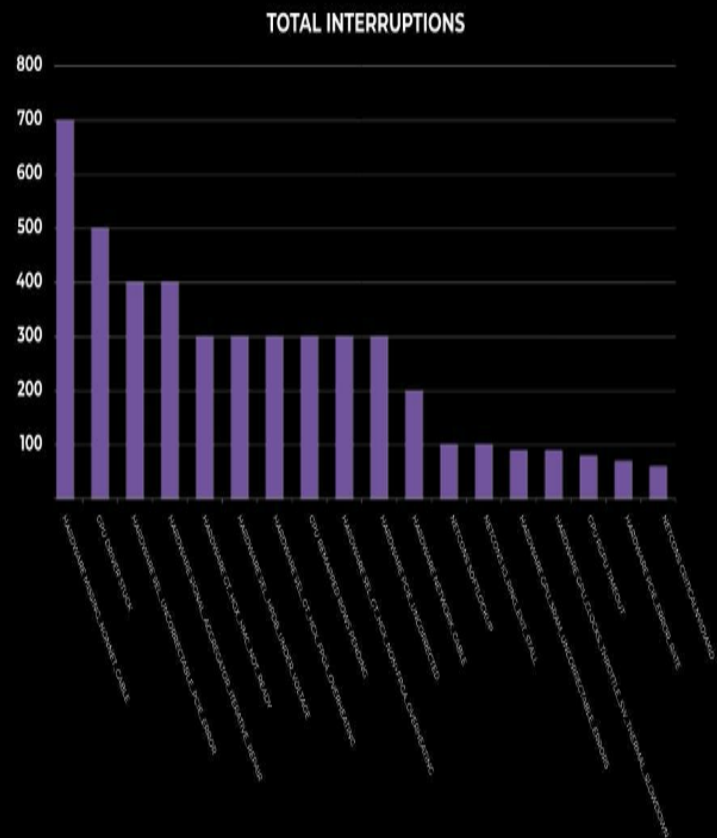


Figure 3-4. In the scaling law,  $N$  represents the number of data points and  $D$  is the number of parameters. Larger models with more data points tend to perform better than larger models with less data. From “First Principles on AI Scaling” by **Dynomight**.

Rather than creating ever-larger monolithic models, we'll see more modular LLMs that can delegate tasks to specialized submodels. Imagine an architecture where the base LLM understands language, but calls on additional “expert” modules trained for niche tasks like legal reasoning, medical diagnostics, or creative writing. Instead of having one large base model trying to be an expert in everything, we would have models that are experts in different things: a healthcare model, a legal model, a creative-writing model, etc.

A specialist model can outperform a generalist model **and** use fewer resources. This is already happening for math problems and web searches in ChatGPT. Researchers have found that GPT performs below average in graduate math (Frieder et al. 2023) and all GPT models have *knowledge cutoffs*, meaning that they are unaware of events after their cutoff date. Currently, when GPT detects that an user is asking a question about an event that occurred after its training cutoff date, it outsources the question to a different model called **SearchGPT**, then uses the results to provide an answer inside the existing chat. Combining generalist models with specialist models allows LLMs to operate efficiently and with greater depth in specialized areas, reducing the computational overhead while improving output precision.

To support such adaptable massive systems, innovations in distributed computing and parallelization will be key. As Evan Morikawa, who led OpenAI Engineering when ChatGPT was becoming ever more popular, explains in an *interview*, models need to be hosted not on singular clusters but across decentralized networks of nodes. This shift will optimize training times, data latency, and real-time inference, making LLMs vastly more efficient in handling large-scale, real-world applications.

## Hybrid Architectures: Merging Neural Networks with Symbolic AI

One of the current limitations of LLMs lies in their reliance on deep learning alone, which excels at pattern recognition but struggles with symbolic reasoning and logic: in many cases, scientists already discovered patterns and codified them into symbolic formulas (e.g., Newton's Theory of Gravity), but the way neural networks are trained doesn't allow them to use existing formulas—they have to rediscover patterns by themselves.

The future of LLMs will involve hybrid architectures that merge the strengths of neural networks with symbolic AI approaches. These architectures will allow models not only to predict the next word in a sentence but also use known rules and formulas, as humans do.

*Neurosymbolic*, or “hybrid,” AI architectures unite the intuitive, pattern-matching power of neural networks with the precise, rule-based reasoning of symbolic systems. LLMs excel at processing text and generating natural-sounding responses by learning statistical regularities from massive dataset, while symbolic AI can represent explicit facts, logical constraints, and rules, making it far easier to trace the reasoning process and enforce consistency. By merging these two approaches, we can develop systems that can understand human language, perform rigorous logical operations, *and* provide explanations for their conclusions.

In practice, this can manifest in multiple ways. For instance, one method is to have an LLM convert user queries into structured representations—such as logical formulas—and then rely on a symbolic reasoner to apply domain-specific rules or constraints. This hybrid approach can also aid in explainability—one of the key weaknesses in today’s LLMs. Users will be able to query why the model arrived at a particular conclusion, and the model can refer to the symbolic pathways used in the answer, providing a more transparent window into its decision-making process.

## **Sparse and Mixture-of-Experts Models**

One of the biggest bottlenecks in scaling LLMs today is their sheer computational cost. Current models process every input with all their parameters, at inference time, regardless of the complexity or simplicity of the task. Future architectures will move toward sparse models and mixture-of-experts systems, where only a subset of the model’s parameters is activated for a given task. In *sparse* models, only the most relevant parameters or neurons are activated for a particular query, allowing for massive reductions in resource consumption while maintaining high-quality results. We believe that sparse modeling, combined with the modular approach, will lead to the development of powerful, efficient LLMs capable of running on consumer-grade hardware while delivering enterprise-level performance.

*Mixture-of-experts* (MoE) models, by contrast, allow LLMs to dynamically activate specialized “experts” based on the input’s requirements. A user

asking for medical advice would engage a different subset of the model's neurons than a user requesting help with poetry. This approach drastically reduces the number of computations per query while increasing the depth of understanding in each domain. It's a "divide and conquer" strategy, where LLMs focus computational resources only where they are needed most.

## Memory-Augmented Models: Towards Persistent, Context-Rich AI

Current LLMs operate with limited memory. While capable of handling context within a few thousand tokens, they struggle to maintain long-term memory across sessions. The next generation of LLMs will address this with *memory-augmented architectures* capable of storing and retrieving vast amounts of data over long periods. These models will have persistent memory layers, allowing them to recall user interactions from years ago or build a comprehensive knowledge base that evolves with time.

This kind of persistent memory will also revolutionize how LLMs handle personalized tasks (Fountas et al. 2024). Rather than starting from scratch with each interaction, future models will remember the user's preferences, needs, and history, enabling richer, more nuanced conversations and solutions. However, that comes with its own challenges in production, including unexpected behavior when dealing with inconsistencies in data. For example, imagine a single parent who is a senior government official but who also asks questions about how to raise a female child without specifically telling the model that the questions are about another person, like asking "what are the signs that my first period is arriving?" instead of "what are the signs that *my daughter's* first period is arriving?" The personalization algorithm may incorrectly conclude that the user is a teenage girl who is *also* a senior government official.

Persistent memory will be key in enterprise applications, where models will continuously learn from organizational data, building an ever-growing repository of insights and expertise.

## Interpretable and Self-Optimizing Models

As LLMs become more pervasive, the need for interpretability will grow. Users and businesses alike will demand models that can explain their reasoning, mitigate biases, and adapt in real-time. Future LLM architectures will include built-in interpretability features using causal learning, where each decision or prediction can be traced back through a chain of reasoning or probabilistic mapping.

These models will also be self-optimizing. Using reinforcement learning, LLMs will learn from user feedback, fine-tuning their own parameters to better align with desired outcomes. Over time, these models will become more personalized, adjusting not just to individuals but also to the specific needs of organizations or industries. Imagine a legal AI model that, after interacting with a team of lawyers for months, begins to understand the specific nuances of that firm's legal style, approach to risk, and preferred legal precedents. These models will learn through iterative feedback, continuously improving without needing massive retraining efforts. They are currently being explored (Jin et al. 2024; Huang et al. 2022) as agents to aid operational productivity, but so many applications remain unexplored.

## Cross-Model Collaboration and Meta-Learning

In the future, no single LLM will operate in isolation. We'll see architectures where multiple models collaborate, exchanging data, insights, and strategies in real-time.

*Meta-learning* will also become more prominent, in which, instead of being trained from scratch, LLMs learn how to learn. This means they will be capable of adjusting their architectures dynamically based on the tasks they encounter and optimizing themselves without human intervention, using different distillation techniques. This shift will push LLMs towards becoming self-evolving entities, reducing the need for constant retraining and manual updates.

Overall, we will move beyond the brute-force scaling of today's models to more refined, hybridized architectures capable of reasoning, learning, and adapting in ways that feel almost human.

As LLMs increasingly interact with multimodal data (text, images, audio, etc.), multimodal fine-tuning techniques will become essential. These methods will enable LLMs to integrate and process information from various modalities, enhancing their ability to perform complex tasks that require understanding of diverse data types.

## **RAG**

RAG models will continue to evolve, integrating retrieval-based components with generative models to enhance accuracy and relevance. These hybrid models will retrieve relevant information from large databases or knowledge sources and use it to generate more informed and contextually appropriate responses. Advances in real-time retrieval mechanisms will allow them to access and utilize up-to-date information dynamically, so they can provide current and contextually relevant responses. This promises to improve RAG models' effectiveness in applications such as customer support, knowledge management, and content creation.

Future RAG systems will better integrate with knowledge graphs and external databases, enabling LLMs to leverage structured knowledge for more accurate, detailed, factually correct, and comprehensive responses, even to complex queries.

Innovations in retrieval mechanisms will focus on improving efficiency and scalability (Gao et al. 2024). Techniques like approximate nearest neighbor search and indexing will be optimized to handle large-scale data and reduce latency in retrieval processes.

To conclude, sparse models and modular frameworks will make LLMs far more efficient, while memory-augmented models will bring persistence and depth to their understanding. The future of LLMs is not just one of bigger models but smarter ones—architectures that grow, learn, and reason,

forming a new foundation for AI-driven innovation across every aspect of society.

## **The Future of LLMOps**

We expect the coming decade to bring a lot of innovation to LLMOps, which is only one aspect, but the infrastructure layer that will be guided by that framework. One of the biggest contributions of any Ops framework is helping practitioners understand what tools they can build to automate and streamline best practices across the industry. For example, the biggest contribution of DevOps was the boom in cloud-services infrastructures. For MLOps, it was data and model-versioning tools. For LLMOps, we personally believe that the biggest booms will be in tools for resource optimization, evaluation, and multimodal data management.

## **Advances in GPU Technology**

GPUs play a key role in LLMOps, and their evolution will continue to drive advancements in model performance and efficiency. Over the next decade, we expect that several emerging trends in GPU technology will significantly impact LLMOps.

The future of GPUs will see the rise of highly specialized AI hardware designed specifically for the unique demands of LLM training and inference, as Figure 10-5 illustrates. Companies like NVIDIA, AMD, and Intel are developing next-generation GPUs with enhanced architectures tailored for AI workloads. These include more GPU cores, increased memory bandwidth, and optimized tensor operations to accelerate model training and reduce latency during inference.

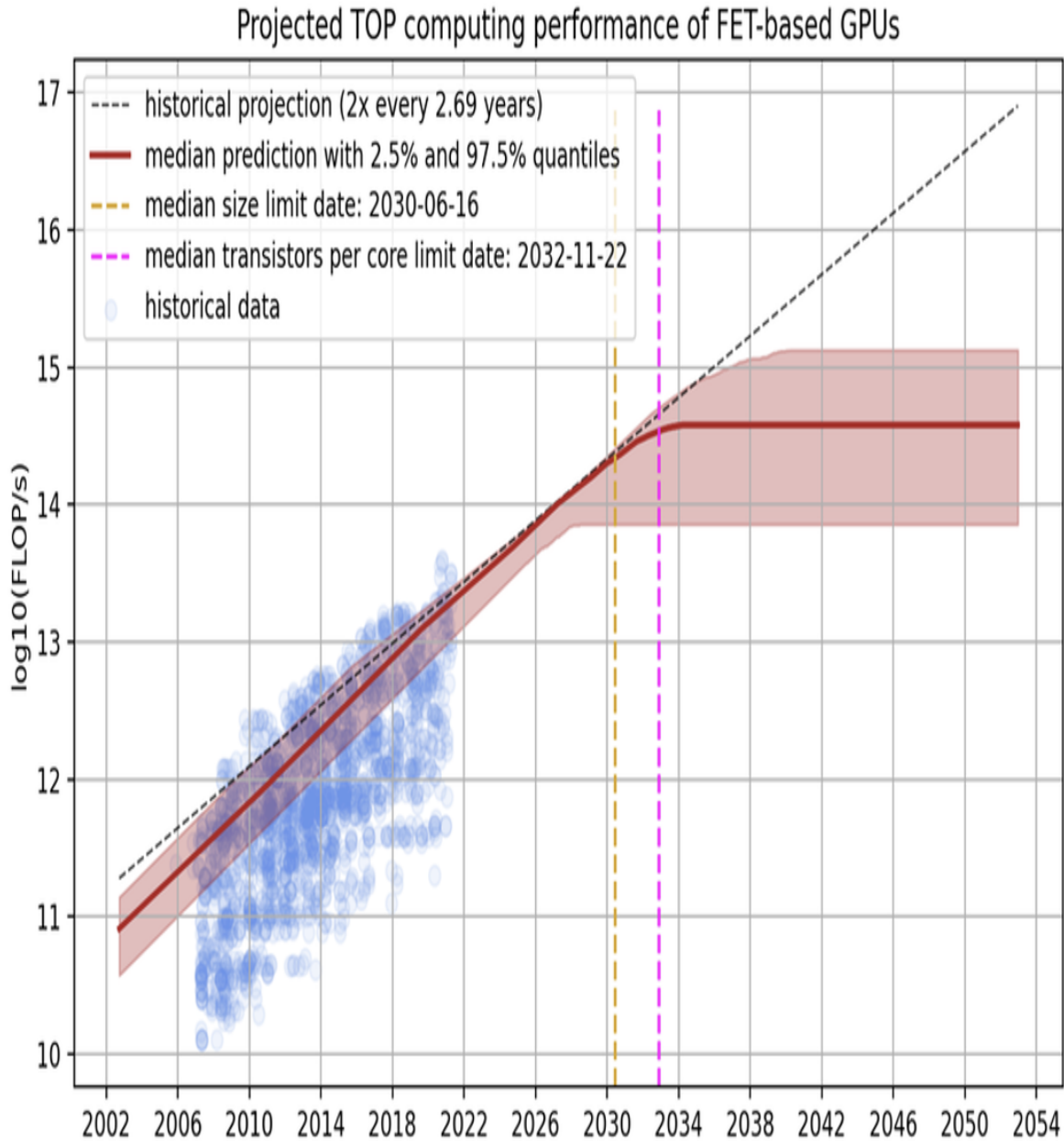


Figure 3-5. Predicting GPU performance in the next 30 years. (source: *Epoch AI*, used with permission)

As LLMs grow in size, the need for efficient multi-GPU and distributed training strategies will become more pronounced. Advances in distributed computing frameworks, such as NVIDIA's NVLink and AMD's Infinity Fabric, will enable more seamless scaling across multiple GPUs and nodes. This will improve training efficiency and reduce the time required to develop and deploy large-scale models.

With the increasing computational demands of LLMs, energy consumption is a critical concern. Future GPUs will focus on enhancing energy efficiency, incorporating techniques like dynamic voltage and frequency scaling (DVFS) and advanced cooling solutions will help mitigate their environmental impact and operational costs.

Finally, although still in its infancy, quantum computing presents potential opportunities for accelerating LLM operations. Quantum processors could complement traditional GPUs, offering exponential speedups for certain types of calculations. Researchers are exploring hybrid approaches that combine quantum and classical computing to tackle complex LLM tasks.

## **Data Management and Efficiency**

As you learned in Chapter 3, effective data management is critical for training and deploying LLMs. Since LLMs require vast amounts of high-quality data, there will be an increasing emphasis on data curation and quality control. We expect techniques for automated data cleaning, augmentation, and validation to become more sophisticated, ensuring that training datasets are diverse, accurate, and representative.

Innovations in data storage and retrieval will be important for managing these massive datasets. Distributed file systems, object storage solutions, and high-performance databases will be incorporated into existing vector databases to handle the scale and complexity of data efficiently.

With growing awareness of data privacy, organizations will adopt advanced methods for protecting user data. Techniques like federated learning and differential privacy will be integrated into data management, allowing LLMs to learn from decentralized data sources without compromising individual privacy.

Synthetically generated data will become a key supplement to real-world data, improving training speed, reducing privacy concerns (as synthetic data is machine-generated) and reducing reliance on expensive or scarce data (Liu et al. 2024; Abdin et al. 2024). The Microsoft Phi-4 small language model released in late 2024 has already achieved good benchmarks at low

cost and a small number of parameters by using some synthetic data (Abdin et al. 2024).

## Privacy and Security

Privacy and security will be paramount as LLMs become more integrated into sensitive and high-stakes applications. The deployment of LLMs will involve advanced security measures to protect against attacks and ensure data integrity. Techniques such as model watermarking, adversarial training, and secure multi-party computation will be employed to safeguard models from tampering and misuse.

As LLMs become more pervasive, ethical considerations will drive the development of guidelines and best practices for their use. New laws like the ones in the US, California and the European Union will incentivize organizations to focus on transparency, fairness, and accountability, implementing measures to ensure that LLMs are used responsibly and ethically. Also, given the massive training and maintenance costs, it's economical for large AI providers to develop models that follow a large market's most restrictive set of rules and deploy them broadly, rather than train and maintain several different models for each set of legal requirements. For example, if the EU requires that models only use anonymized data, it's economical to train and maintain one model worldwide that uses anonymized data rather than two, one that does and one that does not.

## Comprehensive Evaluation Frameworks

We expect that new evaluation frameworks will be developed to assess LLMs across a range of dimensions beyond standard metrics like recall and precision, including factual accuracy, logical accuracy, and coherence. These frameworks will incorporate both qualitative and quantitative measures to provide a holistic view of model performance.

Establishing industry benchmarks and standards, ideally from organizations such as the United Nations International Telecommunications Union (ITU),

the Institute of Electrical and Electronics Engineers Standards Association (IEEE-SA), and the International Standards Organization (ISO), will be essential for comparing LLM performance across different models and platforms. Standardized benchmarks will facilitate fair evaluation and enable organizations to make informed decisions when selecting or developing LLMs.

Ongoing monitoring and evaluation will become standard practice to ensure that LLMs maintain high performance over time. Techniques for continuous evaluation and performance tracking will help identify and address issues as they arise, ensuring that models remain effective and relevant.

## **How to succeed as an LLMOps engineer**

To succeed as an LLMOps engineer, you need to take a system-administrator approach to productionizing LLMs. As Directly Responsible Individuals (DRIs), engineers must be able to understand, evaluate, and manage risk.

Our LLMOps Maturity Model (Chapter 2) can come handy when budgeting for different kinds of errors, from software fault tolerance to model evaluation errors. You cannot monitor everything. Set reasonable expectations, and automate labelling and debugging for different kinds of errors. These could mean keeping an error trail, creating groups of related errors, and using LLM agents to explain and even debug them. An LLMOps engineer often acts as the on-call engineer, which requires dealing with all sorts of problems: hardware, data quality, privacy, user errors. These issues need to be dealt with quickly, often as emergencies. LLMs can help LLMOps engineers sift through the problems and provide suggested solutions, increasing productivity.

As you learned in Chapter 2, LLMOps has four goals: safety, security, robustness and reliability. It can be a tough balancing act to prioritise between monitoring inferences for security testing, optimising model inference pipeline, A/B testing the model releases, managing your compute nodes and optimizing the run pipelines.

Depending on the number of active users the LLM-based application has, the features you are developing for your platform and your team size, LLMOps engineers' workloads can vary massively.

## Conclusion

The world of LLMOps is not just about cutting-edge technology; it's also about processes and measurements. Together, technology, measurements and processes are the pillars supporting the future of AI.

In this book, you've learned about how to apply, deploy, and maintain LLMs efficiently. We discussed the importance of data quality and data management, and explored the art and science of improving LLMs through fine-tuning and prompt engineering.

We've examined the revolutionary potential of RAG to bridge the gap between the general knowledge possessed by LLMs and the recent and/or specialized data some applications need. We've also discussed privacy and security, recognizing that safeguarding our digital interactions is as important as advancing our technological frontiers.

Yet, amid these advancements, it's important to remember the essence of our pursuit. LLMOps is more than a technical discipline; it's about ensuring that our creations serve humanity in ways that are ethical, transparent, and equitable.

Economists recognize AI as one of a very few *general-purpose technologies* (Eloundou et al. 2023), in the same class as the Internet, electricity, or the printing press. These general-purpose technologies tend to be incorporated into almost every human activity, changing the trajectory of progress. LLMOps can help us make the most of AI, preventing and correcting problems and accelerating advances.

The future is ours to shape. Let's make it a future that reflects our highest aspirations and our deepest values.

## References

- Abdin, Marah, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, et al. 2024. “Phi-4 Technical Report.” arXiv. <https://doi.org/10.48550/arXiv.2412.08905>.
- Amodei, Dario. 2024. “Dario Amodei — Machines of Loving Grace.” October 11, 2024. <https://darioamodei.com/machines-of-loving-grace>.
- Bolaños Guerra, Bernardo, and Jorge Luis Morton Gutierrez. 2024. “On Singularity and the Stoics: Why Stoicism Offers a Valuable Approach to Navigating the Risks of AI (Artificial Intelligence).” *AI and Ethics*, August. <https://doi.org/10.1007/s43681-024-00548-w>.
- Chen, Zhuo, Yichi Zhang, Yin Fang, Yuxia Geng, Lingbing Guo, Xiang Chen, Qian Li, et al. 2024. “Knowledge Graphs Meet Multi-Modal Learning: A Comprehensive Survey.” arXiv. <https://doi.org/10.48550/arXiv.2402.05391>.
- Eloundou, Tyna, Sam Manning, Pamela Mishkin, and Daniel Rock. 2023. “GPTs Are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models.” arXiv. <http://arxiv.org/abs/2303.10130>.
- Fountas, Zafeirios, Martin A. Benfeghoul, Adnan Oomerjee, Fenia Christopoulou, Gerasimos Lampouras, Haitham Bou-Ammar, and Jun Wang. 2024. “Human-like Episodic Memory for Infinite Context LLMs.” arXiv. <https://doi.org/10.48550/arXiv.2407.09450>.
- Frieder, Simon, Luca Pinchetti, Alexis Chevalier, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Christian Petersen, and Julius Berner. 2023. “Mathematical Capabilities of ChatGPT.” arXiv. <https://doi.org/10.48550/arXiv.2301.13867>.
- Gao, Yunfan, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. “Retrieval-Augmented Generation for Large Language Models: A Survey.” arXiv. <https://doi.org/10.48550/arXiv.2312.10997>.

Hagendorff, Thilo, Sarah Fabi, and Michal Kosinski. 2023a. “Human-like Intuitive Behavior and Reasoning Biases Emerged in Large Language Models but Disappeared in ChatGPT.” *Nature Computational Science* 3 (10): 833–38. <https://doi.org/10.1038/s43588-023-00527-x>.

———. 2023b. “Thinking Fast and Slow in Large Language Models.” *Nature Computational Science* 3 (10): 833–38. <https://doi.org/10.1038/s43588-023-00527-x>.

“How Meta Trains Large Language Models at Scale.” 2024. *Engineering at Meta* (blog). June 12, 2024. <https://engineering.fb.com/2024/06/12/data-infrastructure/training-large-language-models-at-scale-meta/>.

Huang, Jiaxin, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2022. “Large Language Models Can Self-Improve.” arXiv. <https://doi.org/10.48550/arXiv.2210.11610>.

Ji, Jiaming, Tianyi Qiu, Boyuan Chen, Borong Zhang, Hantao Lou, Kaile Wang, Yawen Duan, et al. 2024. “AI Alignment: A Comprehensive Survey.” arXiv. <https://doi.org/10.48550/arXiv.2310.19852>.

Jin, Haolin, Linghan Huang, Haipeng Cai, Jun Yan, Bo Li, and Huaming Chen. 2024. “From LLMs to LLM-Based Agents for Software Engineering: A Survey of Current, Challenges and Future.” arXiv. <https://doi.org/10.48550/arXiv.2408.02479>.

Liu, Ruibo, Jerry Wei, Fangyu Liu, Chenglei Si, Yanzhe Zhang, Jinmeng Rao, Steven Zheng, et al. 2024. “Best Practices and Lessons Learned on Synthetic Data for Language Models.” arXiv. <https://doi.org/10.48550/arXiv.2404.07503>.

Pan, Shirui, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. “Unifying Large Language Models and Knowledge Graphs: A Roadmap.” *IEEE Transactions on Knowledge and Data Engineering* 36 (7): 3580–99. <https://doi.org/10.1109/TKDE.2024.3352100>.

Papi, Sara, Peter Polak, Ondřej Bojar, and Dominik Macháček. 2024. “How ‘Real’ Is Your Real-Time Simultaneous Speech-to-Text Translation System?” arXiv. <https://doi.org/10.48550/arXiv.2412.18495>.

Tu, Shangqing, Chunyang Li, Jifan Yu, Xiaozhi Wang, Lei Hou, and Juanzi Li. 2024. “ChatLog: Carefully Evaluating the Evolution of ChatGPT Across Time.” arXiv. <https://doi.org/10.48550/arXiv.2304.14106>.

Zhang, Zhehao, Ryan A. Rossi, Branislav Kveton, Yijia Shao, Diyi Yang, Hamed Zamani, Franck Dernoncourt, et al. 2024. “Personalization of Large Language Models: A Survey.” arXiv. <https://doi.org/10.48550/arXiv.2411.00027>.

[OceanofPDF.com](https://oceanofpdf.com)