9

Generative AI in Production

As we've discussed in this book, LLMs have gained significant attention in recent years due to their ability to generate human-like text. From creative writing to conversational chatbots, these generative AI models have diverse applications across industries. However, taking these complex neural network systems from research to real-world deployment comes with significant challenges.

So far, we've talked about models, agents, and LLM apps as well as different use cases, but there are many issues that become important when deploying these apps into production to engage with customers and to make decisions that can have a significant financial impact. This chapter explores the practical considerations and best practices for productionizing generative AI, specifically LLM apps. Before we deploy an application, performance and regulatory requirements need to be ensured, it needs to be robust at scale, and finally monitoring has to be in place. Maintaining rigorous testing, auditing, and ethical safeguards is essential for trustworthy deployment. We'll discuss evaluation and observability, and cover a broad range of topics that encompass the governance and lifecycle management of operationalized AI and decision models, including generative AI models.

While getting an LLM app ready for production, offline evaluation provides a preliminary understanding of a model's abilities in a controlled setting, and when in production, observability offers continuing insights into its performance in live environments. We'll discuss a few tools for either case and I'll give examples. We'll also discuss the deployment of LLM applications and give an overview of available tools and examples for deployment.

Throughout the chapter, we'll work on practical examples with LLM apps, which you can find in the GitHub repository for the book at `https://github.com/benman1/generative_ai_with_langchain`.

The main sections of this chapter are:

- How to get LLM apps ready for production
- How to evaluate LLM apps
- How to deploy LLM apps
- How to observe LLM apps

Let's start with an overview of what it means and involves to get an LLM app ready for production!

How to get LLM apps ready for production

Deploying LLM applications to production is intricate, encompassing robust data management, ethical foresight, efficient resource allocation, diligent monitoring, and alignment with behavioral guidelines. Practices to ensure deployment readiness involve:

- **Data management**: Rigorous attention to data quality is critical to avoid biases that can emanate from imbalanced or inappropriate training data. Substantial efforts in data curation and ongoing scrutiny of model outputs are required to mitigate emerging biases.
- **Ethical deployment and compliance**: LLM applications are potentially capable of generating harmful content, thus necessitating strict review processes, safety guidelines, and compliance with regulations such as HIPAA, especially in sensitive sectors such as healthcare.
- **Resource management**: The resource demands of LLMs call for an infrastructure that is both efficient and environmentally sustainable. Innovation in infrastructure helps to reduce costs and address environmental concerns tied to the energy demands of LLMs.
- **Performance management**: Models must be continually monitored for data drift—where changes in input data patterns can alter model performance—and performance degradation over time. Detecting these deviations necessitates prompt retraining or model adjustments.
- **Interpretability**: To build trust and offer insight into the decision-making processes of LLMs, interpretability tools are increasingly important for users who need clarity on how model decisions are reached.
- **Data security**: Protecting sensitive information within LLM processes is essential for privacy and compliance. Strong encryption measures and stringent access controls bolster security measures.

- **Model behavior standards**: Models must align with ethical guidelines beyond basic functional performance—ensuring outputs are constructive (helpful), innocuous (harmless), and trustworthy (honest). This results in stability and societal acceptance.
- **Hallucination mitigation**: Hallucinations refer to instances where LLMs inadvertently generate or recall sensitive personal information from their training data corpus in the outputs, despite no prompting for such details in the input source—highlighting critical privacy concerns and the need for mitigation strategies.

Essential recommendations for deploying LLM apps encompass an array of practices aimed at mitigating technical challenges, improving performance, and ensuring ethical integrity. First and foremost, it's crucial to develop standardized datasets with relevant benchmarks to test and measure model capabilities, including the detection of regressions and alignment with defined goals.

Metrics should be task-specific to gauge the model's proficiency accurately. There also needs to be a robust framework that includes ethical guidelines, safety protocols, and review processes to prevent the generation and dissemination of harmful content. Human reviewers serve as an essential checkpoint in content validation and bring ethical discernment to AI outputs, ensuring adherence across all contexts.

A forward-thinking UX can foster a transparent relationship with users while reinforcing sensible use. This can include anticipating inaccuracies, such as disclaimers on limitations, attributions, and collecting rich user feedback.

To explain outputs, we should invest in interpretability methods that explain how generative AI models arrive at their decisions. Visualizing attention mechanisms or analyzing feature importance can peel back the layers of complexity, which is particularly crucial for high-stakes industries such as healthcare or finance.

We discussed hallucinations in *Chapter 4*, *Building Capable Assistants*. Mitigation techniques include external retrieval and tool augmentation to provide pertinent context, as we discussed in *Chapter 5*, *Building a Chatbot like ChatGPT*, and *Chapter 6*, *Developing Software with Generative AI*, in particular. There is a danger of models recalling private information, and ongoing advances in methods spanning data filtering, architecture adjustments, and inference techniques show promise in mitigating these problems.

For security, we can strengthen role-based access policies, employ stringent data encryption standards, adopt anonymization best practices where feasible, and ensure continuous verification through compliance audits. Security is a huge topic in the context of LLMs, however, we'll focus on evaluation, observability, and deployment in this chapter.

We need to optimize infrastructure and resource usage by employing distributed techniques such as data parallelism or model parallelism to facilitate workload distribution across multiple processing units. We can employ techniques such as model compression or other computer architectural optimizations for more efficient deployment regarding inference speed and latency management. Techniques such as model quantization, discussed in *Chapter 8*, *Customizing LLMs and Their Output*, or model distillation can also help reduce the resource footprint of the model. Further, storing model outputs can reduce latency and costs for repeated queries.

With insightful planning and preparation, generative AI promises to transform industries from creative writing to customer service. But thoughtfully navigating the complexities of these systems remains critical as they continue to permeate increasingly diverse domains. This chapter aims to provide a practical guide to some of the pieces that we haven't covered in this book so far, aiming to help you build impactful and responsible generative AI applications. We cover strategies for data curation, model development, infrastructure, monitoring, and transparency.

Before we continue our discussion, a few words on terminology are in order. Let's start by introducing MLOps and similar terms, and define what they mean and imply.

Terminology

MLOps is a paradigm that focuses on deploying and maintaining machine learning models in production reliably and efficiently. It combines the practices of DevOps with machine learning to transition algorithms from experimental systems to production systems. MLOps aims to increase automation, improve the quality of production models, and address business and regulatory requirements.

LLMOps is a specialized sub-category of MLOps. It refers to the operational capabilities and infrastructure necessary for fine-tuning and operationalizing LLMs as part of a product. While it may not be drastically different from the concept of MLOps, the distinction lies in the specific requirements connected to handling, refining, and deploying massive language models such as GPT-3, which houses 175 billion parameters.

The term **LMOps** is more inclusive than LLMOps as it encompasses various types of language models, including both LLMs and smaller generative models. This term acknowledges the expanding landscape of language models and their relevance in operational contexts.

Foundational Model Orchestration (**FOMO**) specifically addresses the challenges faced when working with foundation models, that is, models trained on broad data that can be adapted to a wide range of downstream tasks. It highlights the need for managing multi-step processes, integrating with external resources, and coordinating the workflows involving these models.

The term **ModelOps** focuses on the governance and lifecycle management of AI and decision models as they are deployed. Even more broadly, **AgentOps** involves the operational management of LLMs and other AI agents, ensuring their appropriate behavior, managing their environment and resource access, and facilitating interactions between agents while addressing concerns related to unintended outcomes and incompatible objectives.

While FOMO emphasizes the unique challenges of working specifically with foundational models, LMOps provides a more inclusive and comprehensive coverage of a wider range of language models beyond just the foundational ones. LMOps acknowledges the versatility and increasing importance of language models in various operational use cases, while still falling under the broader umbrella of MLOps. Finally, AgentOps explicitly highlights the interactive nature of agents consisting of generative models operating with certain heuristics and includes tools.

The emergence of all of these very specialized terms underscores the rapid evolution of the field; however, their long-term prevalence is unclear. MLOps is widely used and often encompasses the many more specialized terms we just covered. Therefore, we'll stick to MLOps for the remainder of this chapter.

Before productionizing any LLM app, we should first evaluate its output, so we should start with this. We will focus on the evaluation methods provided by LangChain.

How to evaluate LLM apps

The crux of LLM deployment lies in the meticulous curation of training data to preempt biases, implementing human-led annotation for data enhancement, and establishing automated output monitoring systems. Evaluating LLMs either as standalone entities or in conjunction with an agent chain is crucial to ensure they function correctly and produce reliable results, and this is an integral part of the ML lifecycle. The evaluation process determines the performance of the models in terms of effectiveness, reliability, and efficiency.

The goal of evaluating LLMs is to understand their strengths and weaknesses, enhancing accuracy and efficiency while reducing errors, thereby maximizing their usefulness in solving real-world problems. This evaluation process typically occurs offline during the development phase. Offline evaluations provide initial insights into model performance under controlled test conditions and include aspects such as hyperparameter tuning and benchmarking against peer models or established standards. They offer a necessary first step toward refining a model before deployment.

While human assessments are sometimes seen as the gold standard, they are hard to scale and require careful design to avoid bias from subjective preferences or authoritative tones. There are many standardized benchmarks such as MBPP to test basic programming skills, while GSM8K is utilized for multi-step mathematical reasoning. API-Bank evaluates models' aptitudes for making decisions about API calls. ARC puts models' question-answering abilities up against complex integrations of information, whereas HellaSwag assesses common-sense reasoning in physical situations using adversarial filtering.

HumanEval focuses on code generation's functional correctness over syntactic similarity. MMLU assesses language understanding across a wide range of subjects at varying depths, indicating proficiency in specialized domains. SuperGLUE takes GLUE a step further with more challenging tasks to monitor the fairness and comprehension of language models. TruthfulQA brings a unique angle by benchmarking the truthfulness of LLM responses, foregrounding the significance of veracity.

Benchmarks such as MATH demand high-level reasoning capability evaluations. GPT-4's performance on this benchmark varies with prompting method sophistication, from few-shot prompts to reinforcement learning with reward modeling approaches. Notably, dialog-based fine-tuning can sometimes detract from capabilities assessed by measures such as MMLU.

Evaluations can provide insights into how well an LLM generates outputs that are relevant, accurate, and helpful. Tests such as FLAN and FLASK stress behavioral dimensions, thus prioritizing responsible AI systems deployment. This chart compares several open and closed source models on the FLASK benchmark (source: *"FLASK: Fine-grained Language Model Evaluation based on Alignment Skill Sets"* by Ye and colleagues, 2023; `https://arxiv.org/abs/2307.10928`):

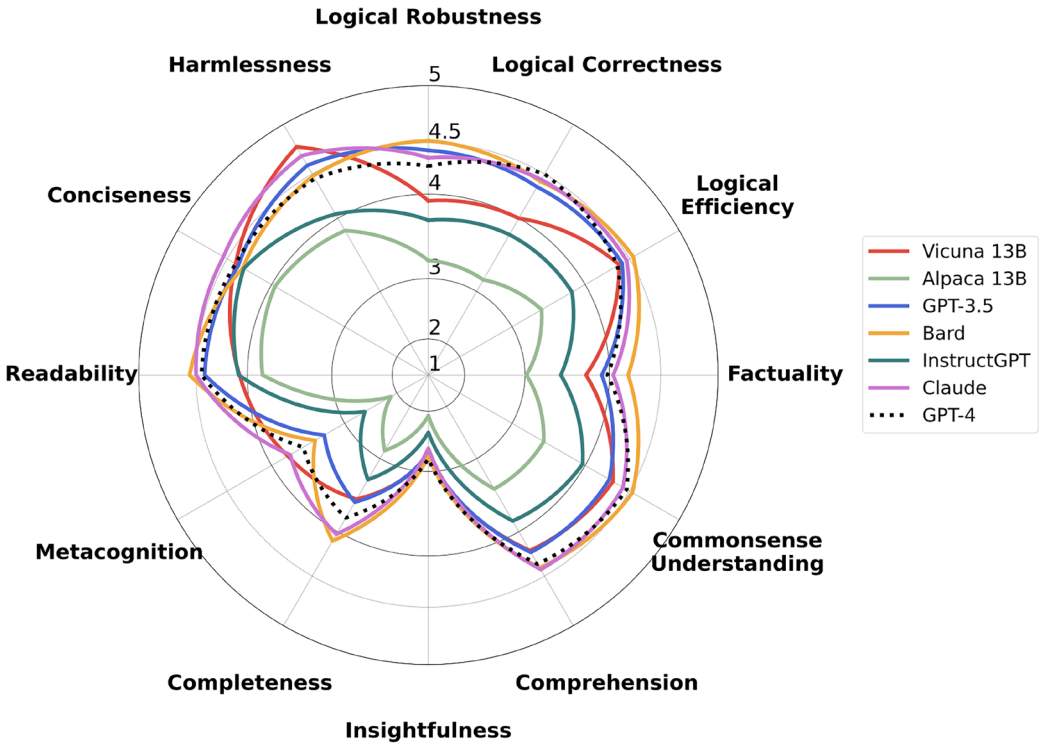

Figure 9.1: Result of an evaluation with Claude as an evaluating language model

In the result reported in the chart, Claude is the LLM evaluating all outputs. This skews results in favor of Claude and models similar to it. Often, GPT-3.5 or GPT-4 are used as evaluators, which shows the OpenAI models emerging as winners.

In LangChain, there are various ways to evaluate the outputs of LLMs, including comparing chain outputs, pairwise string comparisons, string distances, and embedding distances. The evaluation results can be used to determine the preferred model based on the comparison of outputs. Confidence intervals and p-values can also be calculated to assess the reliability of the evaluation results.

LangChain provides several tools for evaluating the outputs of LLMs. A common approach is to compare the outputs of different models or prompts using `PairwiseStringEvaluator`. This prompts an evaluator model to choose between two model outputs for the same input and aggregates the results to determine an overall preferred model.

Other evaluators allow assessing model outputs based on specific criteria such as correctness, relevance, and conciseness. The `CriteriaEvalChain` can score outputs on custom or predefined principles without needing reference labels. Configuring the evaluation model is also possible by specifying a different chat model such as ChatGPT as the evaluator.

You can follow the code in this section online under the `monitoring_and_evaluation` folder in the book's GitHub project. Let's compare outputs of different prompts or LLMs with the `PairwiseStringEvaluator`, which prompts an LLM to select the preferred output given a specific input.

Comparing two outputs

This evaluation requires an evaluator, a dataset of inputs, and two or more LLMs, chains, or agents to compare. The evaluation aggregates the results to determine the preferred model.

The evaluation process involves several steps:

1. **Create the evaluator**: Load the evaluator using the `load_evaluator()` function, specifying the type of evaluator (in this case, `pairwise_string`).
2. **Select the dataset**: Load a dataset of inputs using the `load_dataset()` function.
3. **Define models to compare**: Initialize the LLMs, chains, or agents to compare using the necessary configurations. This involves initializing the language model and any additional tools or agents required.
4. **Generate responses**: Generate outputs for each of the models before evaluating them. This is typically done in batches to improve efficiency.
5. **Evaluate pairs**: Evaluate the results by comparing the outputs of different models for each input. This is often done using a random selection order to reduce positional bias.

Here's an example from the documentation for pairwise string comparisons:

```
from langchain.evaluation import load_evaluator
evaluator = load_evaluator("labeled_pairwise_string")
evaluator.evaluate_string_pairs(
    prediction="there are three dogs",
    prediction_b="4",
    input="how many dogs are in the park?",
    reference="four",
)
```

The output from the evaluator should look as follows:

```
     {'reasoning': "Both assistants provided a direct answer to the user's
question. However, Assistant A's response is incorrect as it stated there
are three dogs in the park, while the reference answer indicates there are
four. On the other hand, Assistant B's response is accurate and matches
the reference answer. Therefore, considering the criteria of correctness
and accuracy, Assistant B's response is superior. \n\nFinal Verdict:
[[B]]",
'value': 'B',
'score': 0

}
```

The evaluation result includes a score between 0 and 1, indicating the effectiveness of each agent, sometimes along with reasoning that outlines the evaluation process and justifies the score. In this specific example against the reference, both results are factually incorrect based on the input. We could remove the reference and let an LLM judge the outputs instead.

Comparing against criteria

LangChain provides several predefined evaluators for different evaluation criteria. These evaluators can be used to assess outputs based on specific rubrics or criteria sets. Some common criteria include conciseness, relevance, correctness, coherence, helpfulness, and controversiality.

`CriteriaEvalChain` allows you to evaluate model outputs against custom or predefined criteria. It provides a way to verify whether an LLM or chain's output complies with a defined set of criteria. You can use this evaluator to assess correctness, relevance, conciseness, and other aspects of the generated outputs.

`CriteriaEvalChain` can be configured to work with or without reference labels. Without reference labels, the evaluator relies on the LLM's predicted answer and scores it based on the specified criteria. With reference labels, the evaluator compares the predicted answer to the reference label and determines its compliance with the criteria.

The evaluation LLM used in LangChain, by default, is GPT-4. However, you can configure the evaluation LLM by specifying other chat models, such as ChatAnthropic or ChatOpenAI, with the desired settings (for example, temperature). The evaluators can be loaded with a custom LLM by passing the LLM object as a parameter to the `load_evaluator()` function.

LangChain supports both custom criteria and predefined principles for evaluation. Custom criteria can be defined using a dictionary of `criterion_name: criterion_description pairs`. These criteria can be used to assess outputs based on specific requirements or rubrics.

Here's a simple example:

```
custom_criteria = {
    "simplicity": "Is the language straightforward and unpretentious?",
    "clarity": "Are the sentences clear and easy to understand?",
    "precision": "Is the writing precise, with no unnecessary words or
details?",
    "truthfulness": "Does the writing feel honest and sincere?",
    "subtext": "Does the writing suggest deeper meanings or themes?",
}
evaluator = load_evaluator("pairwise_string", criteria=custom_criteria)

evaluator.evaluate_string_pairs(
    prediction="Every cheerful household shares a similar rhythm of joy;
but sorrow, in each household, plays a unique, haunting melody.",
    prediction_b="Where one finds a symphony of joy, every domicile of
happiness resounds in harmonious,"
    " identical notes; yet, every abode of despair conducts a dissonant
orchestra, each"
    " playing an elegy of grief that is peculiar and profound to its own
existence.",
    input="Write some prose about families.",
)
```

We can get a very nuanced comparison of the two outputs, as this result shows:

```
{'reasoning': 'Response A is simple, clear, and precise. It uses
straightforward language to convey a deep and sincere message about
families. The metaphor of music is used effectively to suggest deeper
meanings about the shared joys and unique sorrows of families.\n\nResponse
B, on the other hand, is less simple and clear. The language is more
complex and pretentious, with phrases like "domicile of happiness" and
"abode of despair" instead of the simpler "household" used in Response A.
The message is similar to that of Response A, but it is less effectively
conveyed due to the unnecessary complexity of the language.\n\nTherefore,
based on the criteria of simplicity, clarity, precision, truthfulness,
```

```
and subtext, Response A is the better response.\n\n[[A]]', 'value': 'A',
'score': 1}
```

Alternatively, you can use the predefined principles available in LangChain, such as those from Constitutional AI. These principles are designed to evaluate the ethical, harmful, and sensitive aspects of the outputs. The use of principles in evaluation allows for a more focused assessment of the generated text.

String and semantic comparisons

LangChain supports string comparison and distance metrics for evaluating LLM outputs. String distance metrics such as Levenshtein and Jaro provide a quantitative measure of similarity between predicted and reference strings. Embedding distances using models such as SentenceTransformers calculates semantic similarity between the generated and expected texts.

Embedding distance evaluators can use embedding models, such as those based on GPT-4 or Hugging Face embeddings, to compute vector distances between the predicted and reference strings. This measures the semantic similarity between the two strings and can provide insights into the quality of the generated text.

Here's a quick example from the documentation:

```
from langchain.evaluation import load_evaluator
evaluator = load_evaluator("embedding_distance")
evaluator.evaluate_strings(prediction="I shall go", reference="I shan't
go")
```

The evaluator returns the score 0.0966466944859925. You can change the embeddings used with the `embeddings` parameter in the `load_evaluator()` call.

This often gives better results than older string distance metrics, but these are also available and allow for simple unit testing and assessment of accuracy. String comparison evaluators compare predicted strings against reference strings or inputs.

String distance evaluators use distance metrics, such as the Levenshtein or Jaro distance, to measure the similarity or dissimilarity between predicted and reference strings. This provides a quantitative measure of how similar the predicted string is to the reference string.

Finally, there's an agent trajectory evaluator, where the `evaluate_agent_trajectory()` method is used to evaluate the input, prediction, and agent trajectory.

We can also use LangSmith, a companion project for LangChain that aims to facilitate the passage of LLM apps from prototype to production, to compare our performance against a dataset. Let's step through an example!

Running evaluations against datasets

As we've mentioned, comprehensive benchmarking and evaluation, including testing, are critical for safety, robustness, and intended behavior. We can run evaluations against benchmark datasets in LangSmith as we'll see now. First, please make sure you create an account on LangSmith here: `https://smith.langchain.com/`.

You can obtain an API key and set it as `LANGCHAIN_API_KEY` in your environment. We can also set environment variables for project ID and tracing:

```
import os
os.environ["LANGCHAIN_TRACING_V2"] = "true"
os.environ["LANGCHAIN_PROJECT"] = "My Project"
```

This configures LangChain to log traces. If we don't tell LangChain the project ID, it will log against the `default` project. After this setup, when we run our LangChain agent or chain, we'll be able to see the traces on LangSmith.

Let's log a run!

```
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI()
llm.predict("Hello, world!")
```

We can find all these runs on LangSmith. LangSmith lists all runs so far on the LangSmith project page: `https://smith.langchain.com/projects`

We can also find all runs via the LangSmith API:

```
from langsmith import Client
client = Client()
runs = client.list_runs()
print(runs)
```

We can list runs from a specific project or by `run_type`, for example, `chain`. Each run comes with inputs and outputs, as `runs[0].inputs` and `runs[0].outputs`, respectively.

We can create a dataset from existing agent runs with the `create_example_from_run()` function – or from anything else. Here's how to create a dataset with a set of questions:

```
questions = [
    "A ship's parts are replaced over time until no original parts remain. Is it still the same ship? Why or why not?",  # The Ship of Theseus Paradox
    "If someone lived their whole life chained in a cave seeing only shadows, how would they react if freed and shown the real world?",  # Plato's Allegory of the Cave
    "Is something good because it is natural, or bad because it is unnatural? Why can this be a faulty argument?",  # Appeal to Nature Fallacy
    "If a coin is flipped 8 times and lands on heads each time, what are the odds it will be tails next flip? Explain your reasoning.",  # Gambler's Fallacy
    "Present two choices as the only options when others exist. Is the statement \"You're either with us or against us\" an example of false dilemma? Why?",  # False Dilemma
    "Do people tend to develop a preference for things simply because they are familiar with them? Does this impact reasoning?",  # Mere Exposure Effect
    "Is it surprising that the universe is suitable for intelligent life since if it weren't, no one would be around to observe it?",  # Anthropic Principle
    "If Theseus' ship is restored by replacing each plank, is it still the same ship? What is identity based on?",  # Theseus' Paradox
    "Does doing one thing really mean that a chain of increasingly negative events will follow? Why is this a problematic argument?",  # Slippery Slope Fallacy
    "Is a claim true because it hasn't been proven false? Why could this impede reasoning?",  # Appeal to Ignorance
]
shared_dataset_name = "Reasoning and Bias"
ds = client.create_dataset(
    dataset_name=shared_dataset_name, description="A few reasoning and cognitive bias questions",
```

```
)
for q in questions:
    client.create_example(inputs={"input": q}, dataset_id=ds.id)
```

We can then define an LLM agent or chain on the dataset like this:

```
from langchain.chat_models import ChatOpenAI
from langchain.chains import LLMChain
llm = ChatOpenAI(model="gpt-4", temperature=0.0)
def construct_chain():
    return LLMChain.from_string(
        llm,
        template="Help out as best you can.\nQuestion: {input}\nResponse:
",
    )
```

To run an evaluation on a dataset, we can either specify an LLM or – for parallelism – use a constructor function to initialize the model or LLM app for each input. Now, to evaluate the performance against our dataset, we need to define an evaluator as we saw in the previous section:

```
from langchain.smith import RunEvalConfig
evaluation_config = RunEvalConfig(
    evaluators=[
        RunEvalConfig.Criteria({"helpfulness": "Is the response
helpful?"}),
        RunEvalConfig.Criteria({"insightful": "Is the response carefully
thought out?"})
    ]
)
```

As seen, the criteria are defined by a dictionary that includes a criterion as a key and a question to check for as the value.

We'll pass a dataset together with the evaluation configuration with evaluators to `run_on_dataset()` to generate metrics and feedback:

```
from langchain.smith import run_on_dataset
results = run_on_dataset(
  client=client,
  dataset_name=shared_dataset_name,
  dataset=dataset,
```

```
    llm_or_chain_factory=construct_chain,
    evaluation=evaluation_config
)
```

Similarly, we could pass a dataset and evaluators to `run_on_dataset()` to generate metrics and feedback asynchronously.

We can view the evaluator feedback in the LangSmith UI to identify areas for improvement:

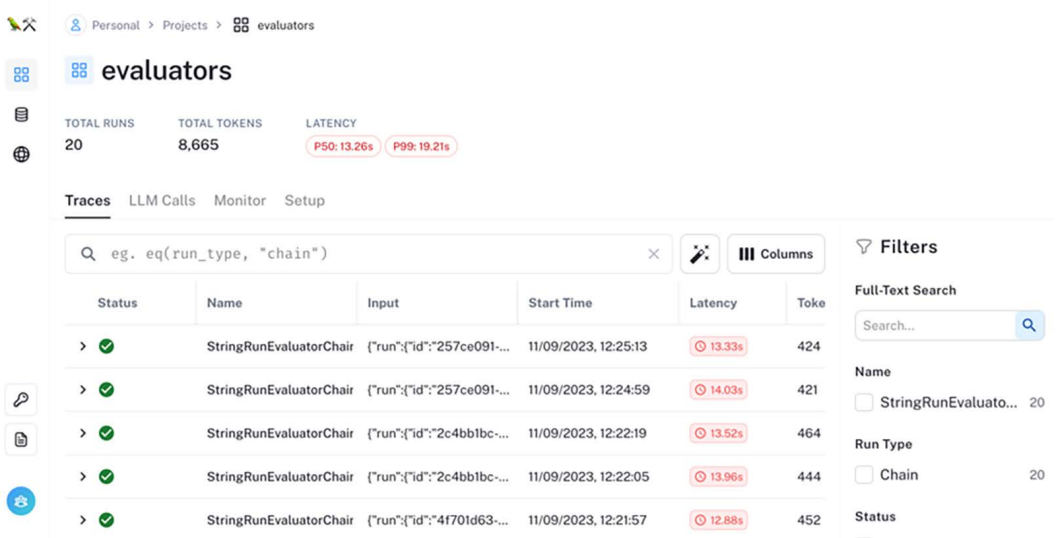

Figure 9.2: Evaluators in LangSmith

We can click on any of these evaluations to see some detail, for example, for the careful thinking evaluator, we get this prompt that includes the original answer from the LLM:

```
You are assessing a submitted answer on a given task or input based on a
set of criteria. Here is the data:
[BEGIN DATA]
***
[Input]: Is something good because it is natural, or bad because it is
unnatural? Why can this be a faulty argument?
***
[Submission]: The argument that something is good because it is natural,
or bad because it is unnatural, is often referred to as the "appeal to
nature" fallacy. This argument is faulty because it assumes that what is
natural is automatically good or beneficial, and what is unnatural is
automatically bad or harmful. However, this is not always the case. For
```

```
example, many natural substances can be harmful or deadly, such as certain
plants or animals. Conversely, many unnatural things, such as modern
medicine or technology, can greatly benefit our lives. Therefore, whether
something is natural or unnatural is not a reliable indicator of its value
or harm.
***
[Criteria]: insightful: Is the response carefully thought out?
***
[END DATA]
Does the submission meet the Criteria? First, write out in a step by step
manner your reasoning about each criterion to be sure that your conclusion
is correct. Avoid simply stating the correct answers at the outset. Then
print only the single character "Y" or "N" (without quotes or punctuation)
on its own line corresponding to the correct answer of whether the
submission meets all criteria. At the end, repeat just the letter again by
itself on a new line.
```

We get this evaluation:

```
The criterion is whether the response is insightful and carefully thought
out.

The submission provides a clear and concise explanation of the "appeal to
nature" fallacy, demonstrating an understanding of the concept. It also
provides examples to illustrate why this argument can be faulty, showing
that the respondent has thought about the question in depth. The response
is not just a simple yes or no, but a detailed explanation that shows
careful consideration of the question.

Therefore, the submission does meet the criterion of being insightful and
carefully thought out.
```

A way to improve performance for a few types of problems is to use few-shot prompting. LangSmith can help us with this as well. You can find more examples of this in the LangSmith documentation.

We haven't discussed data annotation queues, a new feature in LangSmith that addresses a critical gap that emerges after prototyping. Each log can be filtered by attributes such as errors to focus on problematic cases, or manually reviewed and annotated with labels or feedback and edited as needed. Edited logs can be added to a dataset for uses including fine-tuning the model.

This concludes the topic of evaluation here. Now that we've evaluated our agent, let's say we are happy with the performance and have decided to deploy it! What should we do next?

How to deploy LLM apps

Given the increasing use of LLMs in various sectors, it's imperative to understand how to effectively deploy models and apps into production. Deployment services and frameworks can help to scale the technical hurdles. There are lots of different ways to productionize LLM apps or applications with generative AI.

Deployment for production requires research into, and knowledge of, the generative AI ecosystem, which encompasses different aspects including:

- **Models and LLM-as-a-Service**: LLMs and other models either run on-premises or offered as an API on vendor-provided infrastructure.
- **Reasoning heuristics**: Retrieval Augmented Generation (RAG), Tree-of-Thought, and others.
- **Vector databases**: Aid in retrieving contextually relevant information for prompts.
- **Prompt engineering tools**: These facilitate in-context learning without requiring expensive fine-tuning or sensitive data.
- **Pre-training and fine-tuning**: For models specialized for specific tasks or domains.
- **Prompt logging, testing, and analytics**: An emerging sector inspired by the desire to understand and improve the performance of LLMs.
- **Custom LLM stack**: A set of tools for shaping and deploying solutions built on LLMs.

We discussed models in *Chapter 1*, *What Is Generative AI?* and *Chapter 3*, *Getting Started with LangChain*, reasoning heuristics in *Chapter 4*, *Building Capable Assistants - Chapter 7*, *LLMs for Data Science*, vector databases in *Chapter 5*, *Building a Chatbot like ChatGPT*, and prompts and fine-tuning in *Chapter 8*, *Customizing LLMs and Their Output*. In the present chapter, we'll focus on logging, monitoring, and custom tools for deployment.

LLMs are typically utilized using external LLM providers or self-hosted models. With external providers, computational burdens are shouldered by companies such as OpenAI or Anthropic, while LangChain facilitates business logic implementation. However, self-hosting open-source LLMs can significantly decrease costs, latency, and privacy concerns.

Some tools with infrastructure offer the full package. For example, you can deploy LangChain agents with Chainlit, creating ChatGPT-like UIs with Chainlit. Key features include intermediary step visualization, element management and display (images, text, carousel, and others), and cloud deployment. BentoML is a framework that enables the containerization of machine learning applications to use them as microservices running and scaling independently with automatic generation of OpenAPI and gRPC endpoints.

You can also deploy LangChain to different cloud service endpoints, for example, an Azure Machine Learning online endpoint. With Steamship, LangChain developers can rapidly deploy their apps, with features including production-ready endpoints, horizontal scaling across dependencies, persistent storage of app state, and multi-tenancy support.

LangChain AI, the company maintaining LangChain, is developing a new library called LangServe. Built on top of FastAPI and Pydantic, it streamlines documentation and deployment. Deployment is further facilitated through integration with platforms including GCP's Cloud Run and Replit, allowing quick cloning from an existing GitHub repository. Additional deployment instructions for other platforms will follow shortly based on user input.

The following table summarizes the services and frameworks available for deploying LLM applications:

Name	Description	Type
Streamlit	Open-source Python framework for building and deploying web apps	Framework
Gradio	Lets you wrap models in an interface and host on Hugging Face	Framework
Chainlit	Build and deploy conversational ChatGPT-like apps	Framework
Apache Beam	Tool for defining and orchestrating data processing workflows	Framework
Vercel	Platform for deploying and scaling web apps	Cloud service
FastAPI	Python web framework for building APIs	Framework
Fly.io	App hosting platform with autoscaling and global CDN	Cloud service
DigitalOcean App Platform	Platform to build, deploy, and scale apps	Cloud service
Google Cloud	Services such as Cloud Run to host and scale containerized apps	Cloud service

Steamship	ML infrastructure platform for deploying and scaling models	Cloud service
Langchain-Serve	Tool to serve LangChain agents as web APIs	Framework
BentoML	Framework for model serving, packaging, and deployment	Framework
OpenLLM	Provides open APIs to commercial LLMs	Cloud service
Databutton	No-code platform to build and deploy model workflows	Framework
Azure ML	Managed MLOps service on Azure for models	Cloud service
LangServe	Built on top of FastAPI, but specialized for LLM app deployment	Framework

Table 9.1: Services and frameworks for deploying LLM applications

All of these are well documented with different use cases, often directly referencing LLMs. We've already shown examples with Streamlit and Gradio, and we've discussed how to deploy them to the Hugging Face Hub as an example.

There are a few main requirements for running LLM applications:

- Scalable infrastructure to handle computationally intensive models and potential spikes in traffic
- Low latency for real-time serving of model outputs
- Persistent storage for managing long conversations and app state
- APIs for integration into end-user applications
- Monitoring and logging to track metrics and model behavior

Maintaining cost efficiency can be challenging with large volumes of user interactions and the high costs associated with LLM services. Strategies to manage efficiency include self-hosting models, auto-scaling resource allocations based on traffic, using spot instances, independent scaling, and batching requests to better utilize GPU resources.

The choice of tools and the infrastructure determines trade-offs between these requirements. Flexibility and ease is very important, because we want to be able to iterate rapidly, which is vital due to the dynamic nature of ML and LLM landscapes. It's crucial to avoid getting tied to one solution. A flexible, scalable serving layer that accommodates various models is key. Model composition and cloud providers' selection forms part of this flexibility equation.

For the greatest degree of flexibility, **Infrastructure as Code (IaC)** tools such as Terraform, CloudFormation, or Kubernetes YAML files can recreate your infrastructure reliably and quickly. Moreover, **continuous integration and continuous delivery (CI/CD)** pipelines can automate testing and deployment processes to reduce errors and facilitate quicker feedback and iteration.

Designing a robust LLM application service can be a complex task requiring an understanding of the trade-offs and critical considerations when evaluating serving frameworks. Leveraging one of these solutions for deployment allows developers to focus on developing impactful AI applications rather than infrastructure.

As mentioned, LangChain plays nicely with several open-source projects and frameworks such as Ray Serve, BentoML, OpenLLM, Modal, and Jina. In the next sections, we'll deploy apps using different tools. We'll start with a chat service web server based on FastAPI.

FastAPI web server

FastAPI is a very popular choice for the deployment of web servers. Designed to be fast, easy to use, and efficient, it is a modern, high-performance web framework for building APIs with Python. Lanarky is a small, open-source library for deploying LLM applications that provides convenient wrappers around Flask API as well as Gradio for the deployment of LLM applications. This means you can get a REST API endpoint as well as the in-browser visualization at once and you only need a few lines of code.

A **Representational State Transfer Application Programming Interface (REST API)** is a set of rules and protocols that allows different software applications to communicate with each other over the internet. It follows the principles of REST, which is an architectural style for designing networked applications. A REST API uses HTTP methods (such as GET, POST, PUT, or DELETE) to perform operations on resources, and it typically sends and receives data in a standardized format, such as JSON or XML.

In the library documentation, there are several examples, including a Retrieval QA with Sources Chain, a Conversational Retrieval app, and a Zero Shot agent. Following another example, we'll implement a chatbot web server with Lanarky.

We'll set up a web server using Lanarky that creates a `ConversationChain` instance with an LLM model and settings, and defines routes for handling HTTP requests. The full code for this recipe is available here: `https://github.com/benman1/generative_ai_with_langchain/tree/main/webserver`

First, we'll import the necessary dependencies, including FastAPI for creating the web server and `ConversationChain` and `ChatOpenAI` from LangChain for handling LLM conversations, along with some other required modules:

```
from fastapi import FastAPI
from langchain import ConversationChain
from langchain.chat_models import ChatOpenAI

from lanarky import LangchainRouter
from starlette.requests import Request
from starlette.templating import Jinja2Templates
```

Please note that you need to set your environment variables as explained in *Chapter 3*, *Getting Started with LangChain*. We can do this by importing the `setup_environment()` method from the config module as we've seen in many other examples before:

```
from config import set_environment
set_environment()
```

Now we create a FastAPI app, which will take care of most of the routing, except for LangChain specific requests that Lanarky will cover as we'll see later:

```
app = FastAPI()
```

We can create an instance of `ConversationChain`, specifying the LLM model and its settings:

```
chain = ConversationChain(
        llm=ChatOpenAI(
            temperature=0,
            streaming=True,
        ),
        verbose=True,
    )
```

The `templates` variable gets set to a `Jinja2Templates` class, specifying the directory where templates are located for rendering. This specifies how the webpage will be shown, allowing all kinds of customization:

```
templates = Jinja2Templates(directory="webserver/templates")
```

An endpoint for handling HTTP GET requests at the root path (/) is defined using the FastAPI decorator `@app.get`. The function associated with this endpoint returns a template response for rendering the `index.html` template:

```
@app.get("/")
async def get(request: Request):
    return templates.TemplateResponse("index.html", {"request": request})
```

A `router` object is created as a `LangChainRouter` class. This object is responsible for defining and managing the routes associated with the `ConversationChain` instance. We can add additional routes to the router for handling JSON-based chat that even work with WebSocket requests:

```
langchain_router = LangchainRouter(
    langchain_url="/chat", langchain_object=chain, streaming_mode=1
)
langchain_router.add_langchain_api_route(
    "/chat_json", langchain_object=chain, streaming_mode=2
)
langchain_router.add_langchain_api_websocket_route("/ws", langchain_
object=chain)
app.include_router(langchain_router)
```

Now our application knows how to handle requests made to the specified routes defined within the router, directing them to the appropriate functions or handlers for processing.

We will use Uvicorn to run our application. Uvicorn excels in supporting high-performance, asynchronous frameworks such as FastAPI and Starlette. It is known for its ability to handle a large number of concurrent connections and performs well under heavy loads due to its asynchronous nature.

We can run the web server from the terminal like this:

```
uvicorn webserver.chat:app –reload
```

This command starts a web server, which you can view in your browser, at this local address: `http://127.0.0.1:8000`

The reload switch (`--reload`) is particularly handy, because it means the server will be automatically restarted once you've made any changes.

Here's a snapshot of the chatbot application we've just deployed:

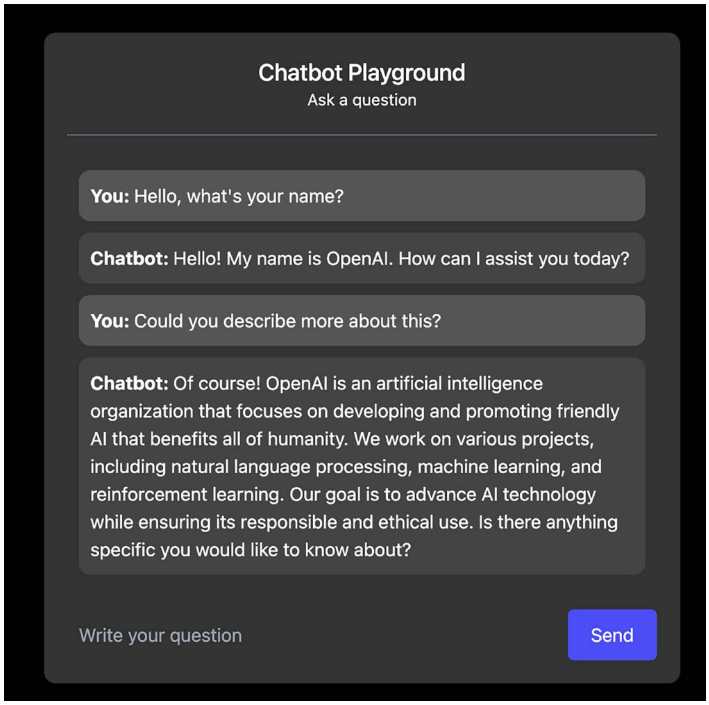

Figure 9.3: Chatbot in Flask/Lanarky

I think this looks quite nice for what little work we've put in. It also comes with a few nice features such as a REST API, a web UI, and a WebSocket interface. While Uvicorn itself does not provide built-in load balancing functionality, it can work together with other tools or technologies such as Nginx or HAProxy to achieve load balancing in a deployment setup, which distributes the incoming client requests across multiple worker processes or instances. The use of Uvicorn with load balancers enables horizontal scaling to handle large traffic volumes, improves response times for clients, and enhances fault tolerance. Finally, Lanarky also plays nicely with Gradio, so with a few extra lines we have this webserver running as a Gradio app up and running.

In the next section, we'll see how to build robust and cost-effective generative AI applications with Ray. We'll build a simple search engine using LangChain for text processing and then use Ray for scaling indexing and serving.

Ray

Ray provides a flexible framework to meet the infrastructure challenges of complex neural networks in production by scaling out generative AI workloads across clusters. Ray helps with common deployment needs such as low-latency serving, distributed training, and large-scale batch inference. Ray also makes it easy to spin up on-demand fine-tuning or scale existing workloads from one machine to many. Its capabilities include:

- Scheduling distributed training jobs across GPU clusters using Ray Train
- Deploying pre-trained models at scale for low-latency serving with Ray Serve
- Running large batch inference in parallel across CPUs and GPUs with Ray Data
- Orchestrating end-to-end generative AI workflows combining training, deployment, and batch processing

We'll use LangChain and Ray to build a simple search engine for the Ray documentation following an example implemented by Waleed Kadous for the anyscale Blog and on the `langchain-ray` repository on GitHub. This can be found here: `https://www.anyscale.com/blog/llm-open-source-search-engine-langchain-ray`

You can see this as an extension of the recipe in *Chapter 5*, *Building a Chatbot like ChatGPT*. You'll also see how to run this as a FastAPI server. The full code for this recipe under semantic search is available here: `https://github.com/benman1/generative_ai_with_langchain/tree/main/search_engine`.

First, we'll ingest and index the Ray docs so we can quickly find relevant passages for a search query:

```
# Load the Ray docs using the LangChain loader
loader = RecursiveUrlLoader("docs.ray.io/en/master/")
docs = loader.load()

# Split docs into sentences using LangChain splitter
chunks = text_splitter.create_documents(
    [doc.page_content for doc in docs],
    metadatas=[doc.metadata for doc in docs])

# Embed sentences into vectors using transformers
embeddings = LocalHuggingFaceEmbeddings('multi-qa-mpnet-base-dot-v1')
```

```
# Index vectors using FAISS via LangChain
db = FAISS.from_documents(chunks, embeddings)
```

This builds our search index by ingesting the docs, splitting them into chunks, embedding the sentences, and indexing the vectors. Alternatively, we can accelerate the indexing by parallelizing the embedding step:

```
# Define shard processing task
@ray.remote(num_gpus=1)
def process_shard(shard):
  embeddings = LocalHuggingFaceEmbeddings('multi-qa-mpnet-base-dot-v1')
  return FAISS.from_documents(shard, embeddings)

# Split chunks into 8 shards
shards = np.array_split(chunks, 8)

# Process shards in parallel
futures = [process_shard.remote(shard) for shard in shards]
results = ray.get(futures)

# Merge index shards
db = results[0]
for result in results[1:]:
  db.merge_from(result)
```

By running embedding on each shard in parallel, we can significantly reduce the indexing time.

We save the database index to disk:

```
db.save_local(FAISS_INDEX_PATH)
```

`FAISS_INDEX_PATH` is an arbitrary file name. I've set it to `faiss_index.db`.

Next, we'll see how we can serve search queries with Ray Serve:

```
# Load index and embedding
db = FAISS.load_local(FAISS_INDEX_PATH)
embedding = LocalHuggingFaceEmbeddings('multi-qa-mpnet-base-dot-v1')

@serve.deployment
```

```
class SearchDeployment:

  def __init__(self):
    self.db = db
    self.embedding = embedding

  def __call__(self, request):
    query_embed = self.embedding(request.query_params["query"])
    results = self.db.max_marginal_relevance_search(query_embed)
    return format_results(results)

deployment = SearchDeployment.bind()

# Start service
serve.run(deployment)
```

This should load the index we generated and lets us serve search queries as a web endpoint!

If we save this to a file called `serve_vector_store.py`, we can get the server up and running using the following command from the `search_engine` directory:

```
PYTHONPATH=../ python serve_vector_store.py
```

Running this command in the terminal gives me this output:

```
Started a local Ray instance.
View the dashboard at 127.0.0.1:8265
```

The message shows us the URL of the dashboard, which we can access in the browser. The search server, however, is running on localhost on port 8080. We can query it from Python:

```
import requests

query = "What are the different components of Ray"
         " and how can they help with large language models (LLMs)?"
response = requests.post("http://localhost:8000/", params={"query":
query})
print(response.text)
```

For me, the server fetches the Ray use cases page at: `https://docs.ray.io/en/latest/ray-overview/use-cases.html`

What I really liked was the monitoring with the Ray Dashboard, which looks like this:

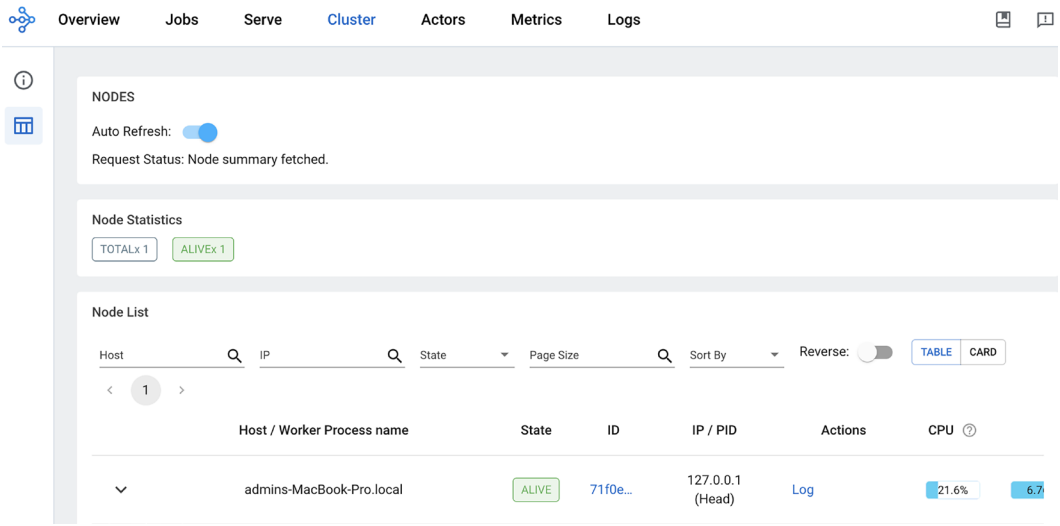

Figure 9.4: Ray Dashboard

This dashboard is very powerful as it can give you a whole bunch of metrics and other information. Collecting metrics is easy, since all you must do is set up and update variables of type `Counter`, `Gauge`, `Histogram`, and others within the deployment object or actor. For time series charts, you should have either Prometheus or the Grafana server installed.

This practical guide has taken you through the key steps of deploying an LLM application locally using LangChain and Ray. We first ingested and indexed documents to power a semantic search engine over the Ray documentation. By leveraging Ray's distributed capabilities, we parallelized the intensive embedding task to accelerate the indexing time. We then served the search application via Ray Serve, which provides a flexible framework for low-latency querying. The Ray dashboard offered helpful monitoring insights into metrics such as request rates, latencies, and errors.

As you can see in the full implementation on GitHub, we can also spin this up as a FastAPI server. This concludes our simple semantic search engine with LangChain and Ray.

As models and LLM apps grow more sophisticated and highly interwoven into the fabric of business applications, observability and monitoring during production become necessary to ensure their accuracy, efficiency, and reliability is ongoing. The next section focuses on the significance of monitoring LLMs and highlights key metrics to track for a comprehensive monitoring strategy.

How to observe LLM apps

The dynamic nature of real-world operations means that the conditions assessed during offline evaluations hardly cover all potential scenarios that LLMs may encounter in production systems. Thus comes the need for observability in production – a more continuous, real-time observation to capture anomalies that offline tests could not anticipate.

We need to implement monitoring tools to track vital metrics regularly. This includes user activity, response times, traffic volumes, financial expenditures, model behavior patterns, and overall satisfaction with the app. Ongoing surveillance allows for the early detection of anomalies such as data drift or unexpected lapses in capabilities.

Observability allows monitoring behaviors and outcomes as the model interacts with actual input data and users in production. It includes logging, tracking, tracing, and alerting mechanisms to ensure healthy system functioning, performance optimization, and catching issues such as model drift early.

Tracking, tracing, and monitoring are three important concepts in the field of software operation and management. While all related to understanding and improving a system's performance, they each have distinct roles. While tracking and tracing are about keeping detailed historical records for analysis and debugging, monitoring is aimed at real-time observation and immediate awareness of issues to ensure optimal system functionality at all times. All three of these concepts fall within the category of observability.

Monitoring is the ongoing process of overseeing the performance of a system or application. This might involve continuously collecting and analyzing metrics related to system health such as memory usage, CPU utilization, network latency, and the overall application/service performance (such as response time). Effective monitoring includes setting up alert systems for anomalies or unexpected behaviors – sending notifications when certain thresholds are exceeded. While tracking and tracing are about keeping detailed historical records for analysis and debugging, monitoring is aimed at real-time observation and immediate awareness of issues to ensure optimal system functionality at all times.

The chief aim for monitoring and observability is to provide insights into LLM app performance and behavior through real-time data. This helps to do the following:

- **Preventing model drift**: LLM performance can degrade over time due to changes in the characteristics of input data or user behavior. Regular monitoring can identify such situations early and apply corrective measures.
- **Performance optimization**: By tracking metrics such as inference times, resource usage, and throughput, you can make adjustments to improve the efficiency and effectiveness of LLM apps in production.
- **A/B testing**: Helps compare how slight differences in models may result in different outcomes, which aids decision-making for model improvements.
- **Debugging issues**: Monitoring helps identify unforeseen problems that can occur during runtime, enabling rapid resolution.
- **Avoiding hallucinations**: We want to ensure the factual accuracy of the response, and – if we are using RAG – retrieved context quality, and sufficient effectiveness in using the context.
- **Ensuring appropriate behavior**: Responses should be relevant, complete, helpful, harmless, conform to the required format, and follow the user's intent.

Since there are so many ways to monitor, it's important to come up with a monitoring strategy. Some things you should consider when coming up with a strategy are:

- **Metrics to monitor**: Define key metrics of interest such as prediction accuracy, latency, throughput, and others based on the desired model performance.
- **Monitoring frequency**: Frequency should be determined based on how critical the model is to operations – a highly critical model may require near real-time monitoring.
- **Logging**: Logs should provide comprehensive details regarding every relevant action performed by the LLM so analysts can track down any anomalies.
- **Alerting mechanism**: The system should raise alerts if it detects anomalous behavior or drastic performance drops.

Monitoring LLMs and LLM apps in production serves multiple purposes, including assessing model performance, detecting abnormalities or issues, optimizing resource utilization, and ensuring consistent and high-quality outputs. By continuously evaluating the behavior and performance of LLM apps via validation, shadow launches, and interpretation along with dependable offline evaluation, organizations can identify and mitigate potential risks, maintain user trust, and provide an optimal experience.

When monitoring LLMs and LLM applications, organizations can rely on a diverse set of metrics to gauge different aspects of performance and user experience. Beyond the crucial metrics of tonality, toxicity, and harmlessness, here is an expanded list that captures a wider range of evaluation areas:

- **Inference latency**: Measures the time it takes for the LLM app to process a request and generate a response. Lower latency ensures a faster and more responsive user experience.
- **Query per Second (QPS)**: Calculates the number of queries or requests that the LLM can handle within a given time frame. Monitoring QPS helps assess scalability and capacity planning.
- **Token per Second (TPS)**: Tracks the rate at which the LLM app generates tokens. TPS metrics are useful for estimating computational resource requirements and understanding model efficiency.
- **Token usage**: The number of tokens correlates with the resource usage such as hardware utilization, latency, and costs.
- **Error rate**: Monitors the occurrence of errors or failures in LLM app responses, ensuring error rates are kept within acceptable limits to maintain the quality of outputs.
- **Resource utilization**: Measures the consumption of computational resources, such as the CPU, memory, and GPU, to reduce costs and avoid bottlenecks.
- **Model drift**: Detects changes in LLM app behavior over time by comparing its outputs to a baseline or ground truth, ensuring the model remains accurate and aligned with expected outcomes.
- **Out-of-distribution inputs**: Identifies inputs or queries falling outside the intended distribution of the LLM's training data, which can cause unexpected or unreliable responses.
- **User feedback metrics**: Monitors user feedback channels to gather insights on user satisfaction, identify areas for improvement, and validate the effectiveness of the LLM app.
- **User engagement**: We can track how users engage with our app; for example, the frequency and duration of sessions or the usage of specific features.
- **Tool/retrieval usage**: Breakdown of the instances when retrieval and tools are used.

This is just a small selection. This list can easily be extended with many more metrics from **Site Reliability Engineering** (**SRE**) relating to task performance or the behavior of the LLM app.

Data scientists and machine learning engineers should check for staleness, incorrect learning, and bias using model interpretation tools such as LIME and SHAP. The most predictive features changing suddenly could indicate a data leak.

Offline metrics such as AUC do not always correlate with online impacts on conversion rate, so it is important to find dependable offline metrics that translate to online gains relevant to the business, ideally direct metrics such as clicks and purchases that the system impacts directly.

Effective monitoring enables the successful deployment and utilization of LLMs, boosting confidence in their capabilities and fostering user trust. It should be cautioned, however, that you should study service providers' privacy and data protection policies when relying on cloud service platforms.

The full code for the recipes in this section are available on GitHub in the `monitoring_and_evaluation` directory of the repository corresponding to this book.

In the next section, we'll start our journey into observability by monitoring the trajectory of an agent.

Tracking responses

Tracking in this context refers to recording the full provenance of responses, including the tools, retrievals, the included data, and the LLM used in generating the output. This is key for auditing and reproducibility of responses. We'll use the terms tracking and tracing interchangeably in this section.

Tracking generally refers to the process of recording and managing information about a particular operation or series of operations within an application or system. For example, in machine learning applications or projects, tracking can involve keeping a record of parameters, hyperparameters, metrics, and outcomes across different experiments or runs. It provides a way to document progress and changes over time.

Tracing is a more specialized form of tracking. It involves recording the execution flow through software/systems. Particularly in distributed systems where a single transaction might span multiple services, tracing helps in maintaining an audit or breadcrumb trail, a detailed source of information about that request path through the system. This granular view enables developers to understand the interaction between various microservices and troubleshoot issues such as latency or failures by identifying exactly where they occurred in the transaction path.

Tracking the trajectory of agents can be challenging due to their broad range of actions and generative capabilities. LangChain comes with functionality for trajectory tracking and evaluation, so seeing the traces of an agent via LangChain is really easy! You just have to set the `return_intermediate_steps` parameter to `True` when initializing an agent or an LLM.

Let's define a tool as a function. It's convenient to re-use the function docstring as a description of the tool. The tool first sends a ping to a website address and returns information about packages transmitted and latency or – in the case of an error – the error message:

```
import subprocess
from urllib.parse import urlparse
from pydantic import HttpUrl
from langchain.tools import StructuredTool

def ping(url: HttpUrl, return_error: bool) -> str:
    """Ping the fully specified url. Must include https:// in the url."""
    hostname = urlparse(str(url)).netloc
    completed_process = subprocess.run(
    ["ping", "-c", "1", hostname], capture_output=True, text=True
    )
    output = completed_process.stdout
    if return_error and completed_process.returncode != 0:
        return completed_process.stderr
    return output

ping_tool = StructuredTool.from_function(ping)
```

Now we set up an agent that uses this tool with an LLM to make the calls given a prompt:

```
from langchain.chat_models import ChatOpenAI
from langchain.agents import initialize_agent, AgentType

llm = ChatOpenAI(model="gpt-3.5-turbo-0613", temperature=0)
agent = initialize_agent(
    llm=llm,
    tools=[ping_tool],
    agent=AgentType.OPENAI_MULTI_FUNCTIONS,
    return_intermediate_steps=True, # IMPORTANT!
```

```
)
result = agent("What's the latency like for https://langchain.com?")
```

The agent reports this:

```
The latency for https://langchain.com is 13.773 ms
```

In `results["intermediate_steps"]`, we can see a lot of information about the agent's actions:

```
[(_FunctionsAgentAction(tool='ping', tool_input={'url': 'https://
langchain.com', 'return_error': False}, log="\nInvoking: `ping` with
`{'url': 'https://langchain.com', 'return_error': False}`\n\n\n", message_
log=[AIMessage(content='', additional_kwargs={'function_call': {'name':
'tool_selection', 'arguments': '{\n  "actions": [\n    {\n      "action_
name": "ping",\n      "action": {\n        "url": "https://langchain.
com",\n        "return_error": false\n      }\n    }\n  ]\n}'}},
example=False)]), 'PING langchain.com (35.71.142.77): 56 data bytes\
n64 bytes from 35.71.142.77: icmp_seq=0 ttl=249 time=13.773 ms\
n\n--- langchain.com ping statistics ---\n1 packets transmitted, 1
packets received, 0.0% packet loss\nround-trip min/avg/max/stddev =
13.773/13.773/13.773/0.000 ms\n')]
```

By providing visibility into the system and aiding in problem identification and optimization efforts, this kind of tracking and evaluation can be very helpful.

The LangChain documentation demonstrates how to use a trajectory evaluator to examine the full sequence of actions and responses they generate and grade an OpenAI functions agent. That's potentially very powerful stuff!

Let's have a look beyond LangChain and see what else is out there for observability!

Observability tools

There are quite a few tools available as integrations in LangChain or through callbacks:

- **Argilla**: Argilla is an open-source data curation platform that can integrate user feedback (human-in-the-loop workflows) with prompts and responses to curate datasets for fine-tuning.
- **Portkey**: Portkey adds essential MLOps capabilities like monitoring detailed metrics, tracing chains, caching, and reliability through automatic retries to LangChain.
- **Comet.ml**: Comet offers robust MLOps capabilities for tracking experiments, comparing models and optimizing AI projects.

- **LLMonitor**: Tracks lots of metrics including cost and usage analytics (user tracking), tracing, and evaluation tools (open-source).
- **DeepEval**: Logs default metrics including relevance, bias, and toxicity. Can also help with testing and monitoring model drift or degradation.
- **Aim**: An open-source visualization and debugging platform for ML models. It logs inputs, outputs, and the serialized state of components, enabling visual inspection of individual LangChain executions and comparing multiple executions side by side.
- **Argilla**: An open-source platform for tracking training data, validation accuracy, parameters, and more across machine learning experiments.
- **Splunk**: Splunk's Machine Learning Toolkit can provide observability into your machine learning models in production.
- **ClearML**: An open-source tool for automating training pipelines, seamlessly moving from research to production.
- **IBM Watson OpenScale**: A platform providing insights into AI health with fast problem identification and resolution to help mitigate risks.
- **DataRobot MLOps**: Monitors and manages models to detect issues before they impact performance.
- **Datadog APM integration**: This integration allows you to capture LangChain requests, parameters, prompt completions, and visualize LangChain operations. You can also capture metrics such as request latency, errors, and token/cost usage.
- **Weights and Biases (W&B) tracing**: We've already shown an example of using W&B to monitor fine-training convergence, but it can also fulfill the roles of tracking other metrics and logging and comparing prompts.
- **Langfuse**: With this open-source tool, we can conveniently monitor detailed information along traces regarding the latency, cost, and scores of our LangChain agents and tools.
- **LangKit**: This extracts signals from prompts and responses to ensure safety and security. It currently focuses on text quality, relevance metrics, and sentiment analysis.

There are more tools out there at different stages of maturation. For example, the AgentOps SDK is aiming to provide an interface to a toolkit for evaluating and developing robust and reliable AI agents, but is still in closed alpha.

Most of these integrations are very easy to integrate into LLM pipelines. For example, for W&B, you can enable tracing by setting the `LANGCHAIN_WANDB_TRACING` environment variable to `True`. Alternatively, you can use a context manager with `wandb_tracing_enabled()` to trace a specific block of code. With Langfuse, we can hand over `langfuse.callback.CallbackHandler()` as an argument to the `chain.run()` call.

Some of these tools are open-source, and what's great about these platforms is that they allow full customization and on-premises deployment for use cases where privacy is important. For example, Langfuse is open-source and provides an option of self-hosting. Choose the option that best suits your needs and follow the instructions provided in the LangChain documentation to enable tracing for your agents. Having been released only recently, I am sure there's much more to come for the platform, but it's already great to see traces of how agents execute, detecting loops and latency issues. It enables sharing traces and stats with collaborators to discuss improvements.

Let's have a look at LangSmith now, which is another companion project of LangChain, developed for observability!

LangSmith

LangSmith is a framework for debugging, testing, evaluating, and monitoring LLM applications developed and maintained by LangChain AI, the organization behind LangChain. LangSmith serves as an effective tool for MLOps, specifically for LLMs, by providing features that cover multiple aspects of the MLOps process. It can help developers take their LLM applications from prototype to production by providing features for debugging, monitoring, and optimizing.

LangSmith allows you to:

- Log traces of runs from your LangChain agents, chains, and other components
- Create datasets to benchmark model performance
- Configure AI-assisted evaluators to grade your models
- View metrics, visualizations, and feedback to iterate and improve your LLMs

On the LangSmith web interface, we can get a large set of graphs for a bunch of statistics that can be useful to optimize latency, hardware efficiency, and cost, as we can see here in the monitoring dashboard:

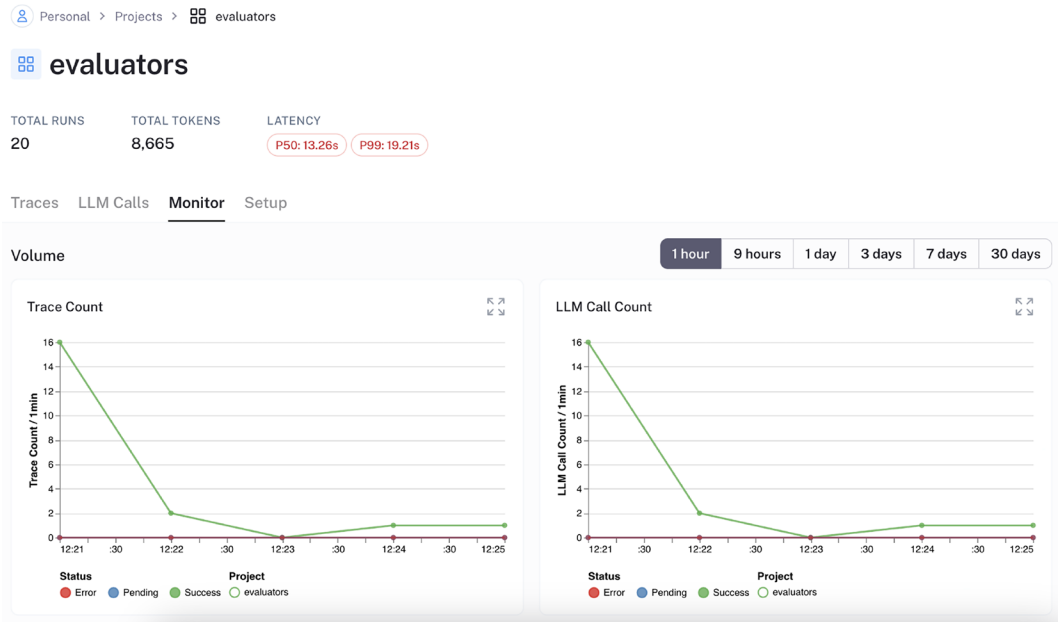

Figure 9.5: Evaluator metrics in LangSmith

The monitoring dashboard includes the following graphs that can be broken down into different time intervals:

Statistics	Category
Trace Count, LLM Call Count, Trace Success Rates, LLM Call Success Rates	Volume
Trace Latency (s), LLM Latency (s), LLM Calls per Trace, Tokens / sec	Latency
Total Tokens, Tokens per Trace, Tokens per LLM Call	Tokens
% Traces w/ Streaming, % LLM Calls w/ Streaming, Trace Time-to-First-Token (ms), LLM Time-to-First-Token (ms)	Streaming

Table 9.2: Statistics in LangSmith

Here's a tracing example in LangSmith for the benchmark dataset run that we saw in the *How to evaluate LLM apps* section:

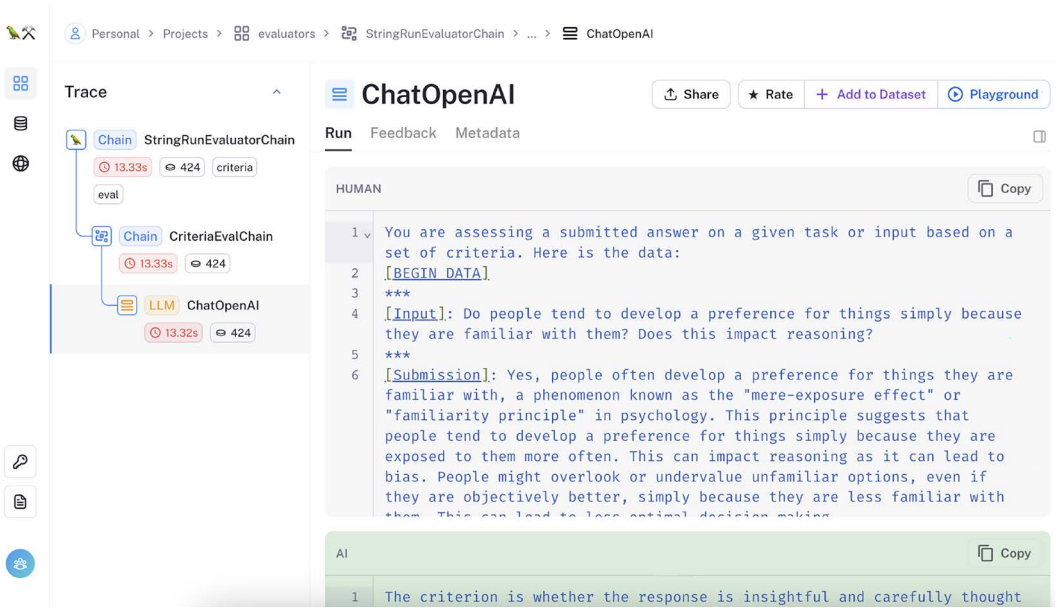

Figure 9.6: Tracing in LangSmith

The platform itself is not open-source, however, LangChain AI, the company behind LangSmith and LangChain, provides some support for self-hosting for organizations with privacy concerns. There are, however, a few alternatives to LangSmith such as Langfuse, Weights and Biases, Datadog APM, Portkey, and PromptWatch, with some overlap in features. We'll focus on LangSmith here because it has a large set of features for evaluation and monitoring, and because it integrates with LangChain.

In the next section, we'll demonstrate the utilization of PromptWatch for prompt tracking of LLMs in production environments.

PromptWatch

PromptWatch records information about response caching, chain execution, prompting and generated output during interactions. The tracing and monitoring can be very useful for debugging and ensuring an audit trail. With PromptWatch.io, you can even track various aspects of LLM chains, actions, retrieved documents, inputs, outputs, execution time, tool details, and more for complete visibility in your system.

Make sure you sign up with PromptWatch.io online and get your API key – you can find it under the account settings.

Let's get the inputs out of the way:

```
from langchain import LLMChain, OpenAI, PromptTemplate
from promptwatch import PromptWatch
```

As discussed in *Chapter 3*, *Getting Started with LangChain*, I've set all API keys in the environment in the `set_environment()` function. If you've followed my recommendation, you can follow the imports up with this:

```
from config import set_environment
set_environment()
```

Otherwise, please make sure you set your environment variables in the way you prefer. Next, we need to set up a prompt and a chain:

```
prompt_template = PromptTemplate.from_template("Finish this sentence
{input}")
my_chain = LLMChain(llm=OpenAI(), prompt=prompt_template)
```

Using the `PromptTemplate` class, the prompt template is configured with one variable, `input`, indicating where the user input should be placed within the prompt.

We can create a `PromptWatch` block, where `LLMChain` is invoked with an input prompt:

```
with PromptWatch() as pw:
    my_chain("The quick brown fox jumped over")
```

This is a simple example of the model generating a response based on the provided prompt. We can see this on PromptWatch.io.

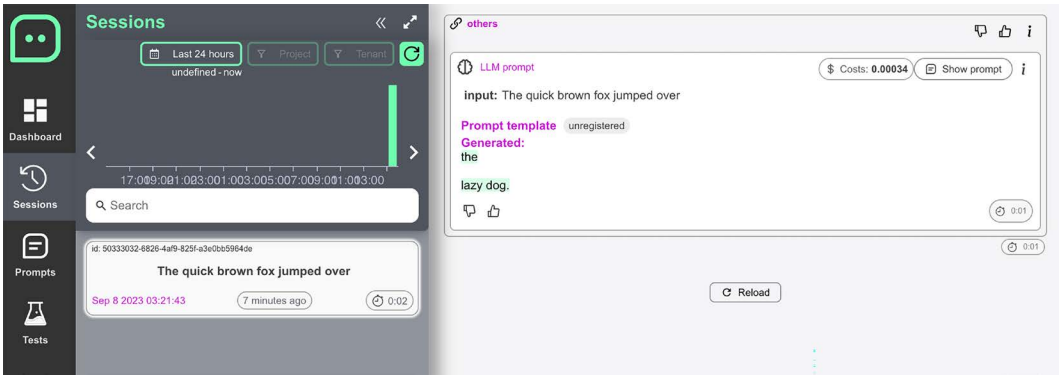

Figure 9.7: Prompt tracking at PromptWatch.io

We can see the prompt together with the LLM's response. We also get a dashboard with a time series of activity, where we can drill down into responses at certain times. This seems quite useful to effectively monitor and analyze prompts, outputs, and costs in real-world scenarios.

The platform allows for in-depth analysis and troubleshooting in the web interface that enables users to identify the root causes of issues and optimize prompt templates. We could have explored more, for example around prompt templates and versioning, but there's only so much we can cover here. `promptwatch.io` can also help with unit testing and versioning prompt templates.

Summary

Taking a trained LLM from research into real-world production involves navigating many complex challenges around aspects such as scalability, monitoring, and unintended behaviors. Responsibly deploying capable, reliable models involves diligent planning around scalability, interpretability, testing, and monitoring. Techniques such as fine-tuning, safety interventions, and defensive design enable us to develop applications that produce helpful, harmless, and readable outputs. With care and preparation, generative AI holds immense potential benefit to industries from medicine to education.

We've delved into deployment and the tools used for deployment. Particularly, we deployed applications with FastAPI and Ray. In earlier chapters, we used Streamlit. There are many more tools we could have explored, for example, the recently emerged LangServe, which is developed with LangChain applications in mind. While it's still relatively fresh, it's definitely worth watching out for more developments in the future.

The evaluation of LLMs is important to assess their performance and quality. LangChain supports comparative evaluation between models, checking outputs against criteria, simple string matching, and semantic similarity metrics. These provide different insights into model quality, accuracy, and appropriate generation. Systematic evaluation is key to ensuring LLMs produce useful, relevant, and sensible outputs.

Monitoring LLMs is a vital aspect of deploying and maintaining these complex systems. With the increasing adoption of LLMs in various applications, ensuring their performance, effectiveness, and reliability is of utmost importance. We've discussed the significance of monitoring LLMs, highlighted key metrics to track for a comprehensive monitoring strategy, and have given examples of how to track metrics in practice.

We've looked at different tools for observability such as PromptWatch and LangSmith. LangSmith provides powerful capabilities to track, benchmark, and optimize LLMs built with LangChain. Its automated evaluators, metrics, and visualizations help accelerate LLM development and validation.

In the next and final chapter, let's discuss what the future of Generative AI will look like.

Questions

Please try and see if you can come up with the answers to these questions from memory. If you are unsure about any of them, you might want to refer to the corresponding section in the chapter:

1. In your opinion, what is the best term for describing the operationalization of language models, LLM apps, or apps that rely on generative models in general?
2. What is a token and why should you know about token usage when querying LLMs?
3. How can we evaluate LLM apps?
4. Which tools can help to evaluate LLM apps?
5. What are the considerations for the production deployment of agents?
6. Name a few tools used for deployment.
7. What are the important metrics for monitoring LLMs in production?
8. How can we monitor LLM applications?
9. What's LangSmith?

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/lang`

10

The Future of Generative Models

In this book, so far, we have discussed generative models for building applications, and we have implemented a few simple ones – for example, for semantic search, applications for content creation, customer service agents, and assistants for developers and data scientists. We have explored techniques such as tool use, agent strategies, semantic search with retrieval augmented generation, and the conditioning of models with prompts and fine-tuning.

In this chapter, we'll deliberate on where this leaves us and where the future leads us. We'll consider weaknesses and socio-technical challenges of generative models, and strategies for mitigation and improvement. We'll focus on value creation opportunities, where unique customization of foundation models for specific use cases stands out. It remains uncertain which entities – big tech firms, start-ups, or foundation model developers – will capture the most upsides. We'll also evaluate and address concerns such as the extinction threat through AI.

Given the massive potential for increased productivity in various industries, venture funding for generative AI start-ups skyrocketed in 2022 and 2023, and major players like Salesforce and Accenture among many others have made big commitments to generative AI with multibillion-dollar investments. We'll discuss potential effects on jobs in multiple industries, and disruptive changes in creative industries, education, law, manufacturing, medicine, and the military.

We will evaluate and address concerns such as misinformation, cybersecurity, privacy, and fairness, and think about how the changes and disruptions brought about by generative AI should influence regulations and practical implementation.

The main sections of this chapter are:

- The current state of generative AI
- Economic consequences
- Societal implications

Let's start with the current state of models and their capabilities.

The current state of generative AI

As discussed in this book, in recent years, generative AI models have attained new milestones in producing human-like content across modalities including text, images, audio, and video. Leading models like OpenAI's GPT-4 and DALL-E 2, and Anthropic's Claude display impressive fluency in content generation, be it textual or creative visual artistry.

Between 2022 and 2023, models have progressed in strides. If generative models were previously capable of producing barely coherent text or grainy images, now we see high-quality 3D images, videos, and the generation of coherent and contextually relevant prose and dialogue, rivaling or even surpassing the fluency levels of humans. These AI models leverage gargantuan datasets and computational scale, enabling them to capture intricate linguistic patterns, display a nuanced understanding of knowledge about the world, translate texts, summarize content, answer natural language questions, create appealing visual art, and acquire the capability to describe images. Seemingly by magic, the AI-generated outputs mimic human ingenuity – painting original art, writing poetry, producing human-level prose, and even engaging in sophisticated aggregation and synthesis of information from diverse sources.

But let's be a bit more nuanced! Generative models come with weaknesses as well as strengths. Deficiencies persist compared to human cognition, including the frequent generation of plausible yet incorrect or nonsensical statements. Hallucinations show a lack of grounding in reality, given that they are based on patterns in data rather than an understanding of the real world. Further, models exhibit difficulties performing mathematical, logical, or causal reasoning. They are easily confused by complex inferential questions, which could limit their applicability in certain fields of work. The black box problem of lack of explainability for outputs as well as for the models themselves hampers troubleshooting efforts, and controlling model behaviors within desired parameters remains challenging. AI can have serious bias issues because of the prejudiced data they are trained on. This can lead to unfair results and make social inequalities worse.

Here is a table summarizing the key strengths and deficiencies of current generative AI compared to human cognition:

Category	**Human Cognition**	**Generative AI Models**
Language Fluency	Contextually relevant, draws meaning from world knowledge	Highly eloquent, reflects linguistic patterns
Knowledge	Conceptual understanding derived from learning and experience	Statistical synthesis lacking grounding
Creativity	Originality reflecting personality and talent	Imaginative but within training distribution
Factual Accuracy	Usually aligns with truth and physical reality	Hallucinations reflecting training data biases
Reasoning	Intuitive yet can apply heuristics after training	Logic is tightly limited to training distribution
Bias	Sometimes recognizes and can override inherent biases	Propagates systemic biases in data
Transparency	Partial, subjective insights from think-aloud techniques	Plausible reasoning from chain-of-thought prompts

Table 10.1: Strengths and deficiencies of LLMs

While LLMs such as GPT-4 showcase language fluency on parity with humans, their lack of grounding, tendency for distortion, opaqueness, and potential for harm underscore deficiencies that temper the promise of generative AI. Progress in domains like logical reasoning and bias mitigation remains at an early stage. As for transparency, while immense complexity poses an immense challenge, determined efforts seek to surface the lineage and mechanisms of reasoning for both humans (advances in the understanding of neurocognition) and AI (interpretability and explainability). Addressing problematic areas is key to developing reliable and trustworthy systems. Throughout the book, we've discussed and implemented potential solutions that address the weaknesses of generative AI.

We should keep in mind, however, that this gap analysis of human versus AI is for highlighting areas of improvement – as we have seen in domains such as Atari games, chess, and Go, AIs can reach superhuman levels if trained properly, and we haven't touched the ceiling yet in many areas. Let's look more broadly at some of the socio-technical challenges involved in unlocking the capabilities of generative AI systems and discuss approaches to overcoming them!

Challenges

The profound potential of generative AI systems indicates an exciting future if development continues at pace. This table shows a summary of a few of the technical and organizational challenges together with approaches to tackle them:

Challenge	Potential Solutions
Knowledge Freshness (+ Concept Drift)	Continuous learning methods like elastic weight consolidation, stream ingestion pipelines, and efficient retraining procedures
Specialized Knowledge	Task-specific demonstrations and prompting, knowledge retrieval and grounding, and context expansion
Downstream Adaptability	Strategic fine-tuning methods, catastrophic forgetting mitigation, and optimized hardware access
Biased Outputs	Bias mitigation algorithms, balanced training data, audits, inclusivity training, and interdisciplinary research
Harmful Content Generation	Moderation systems, interruption and correction, and conditioning methods such as RLHF
Logical Inconsistencies	Hybrid architectures, knowledge bases, and retrieval augmentation
Factual Inaccuracies	Retrieval augmentation, knowledge bases, and consistent knowledge base updating
Lack of Explainability	Model introspection, concept attribution, and interpretable model designs
Privacy Risks	Differential privacy, federated learning, encryption, and anonymization
High Latency and Compute Costs	Model distillation, optimized hardware, and efficient model design
Licensing Limitations	Open/synthetic data, custom data, and fair licensing agreements
Security/Vulnerabilities	Adversarial robustness and cybersecurity best practices

Governance	Compliance frameworks and ethical development governance

Table 10.2: Challenges of generative AI and potential solutions

Challenges of generative AI go beyond just improving content generation—they encompass environmental sustainability, algorithmic equity, and individual privacy. Strategies like employing simplified model architectures, using knowledge distillation, and developing specialized hardware are critical to reducing the carbon footprint of AI in the face of rapid progress. To ensure fair AI, steps such as incorporating balanced datasets, applying bias mitigation algorithms, enforcing fairness through constrained optimization, and promoting inclusivity are essential, despite their complexity.

To counteract potential harm from AI output, such as toxicity or false information (hallucination), techniques like reinforcement learning guided by human feedback and grounding responses in verified knowledge can be employed. Additionally, securing sensitive data through privacy-preserving methods like differential privacy, federated learning, and real-time content correction is fundamental for upholding user dignity.

Finally, staying up to date with the evolving informational landscapes, comprehending specialized domains, and flexibly adapting to emerging needs represent newly visible obstacles as generative models permeate real-world contexts.

Addressing these challenges involves a broad spectrum of responses that must consider the entire life cycle of AI development. Such responses include innovative training objectives focused on consistency, structural knowledge integrations, and design of models for better controllability, as well as software and hardware optimization for infrastructure efficiency.

One of the most effective developments is flexible user control. With concerted effort in research and development, the aim is to steer generative AI toward alignment with societal values. For reasons of computational efficiency and costs, this implies a shift from pretraining to specialized downstream conditioning (particularly, fine-tuning and prompt techniques). This, in turn, will lead to a proliferation of start-ups applying core AI technologies.

Technological innovation together with regulation and transparency of AI development will ensure that generative AI enhances human capability without compromising ethical standards. Looking ahead, generative AI systems are poised to become more powerful and multifaceted.

Let's have a look at some emerging trends in model development!

Trends in model development

The current doubling time in training compute of very large models is about 8 months, outstripping scaling laws such as Moore's Law (transistor density at cost increases at a rate of currently about 18 months) and Rock's Law (costs of hardware like GPUs and TPUs halve every 4 years). This graph illustrates this trend in training compute of large models (source: Epoch, *Parameter, Compute, and Data Trends in Machine Learning*. Retrieved from `https://epochai.org/mlinputs/visualization`):

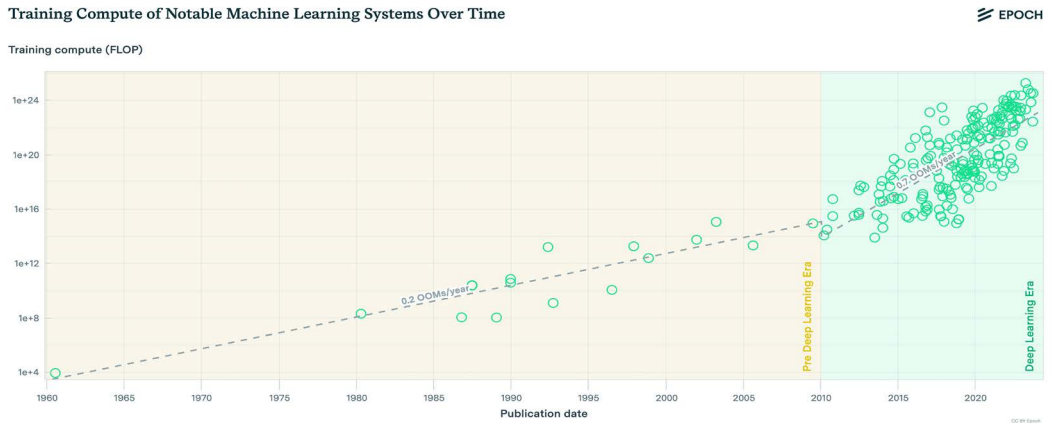

Figure 10.1: Training FLOPs of notable AI systems

The main point from this graph is the increase in compute, which is apparent since the 1960s, and the Cambrian explosion of models of the deep learning era at the top right. As discussed in *Chapter 1*, *What Is Generative AI?*, parameter sizes for large systems have been increasing at a similar rate as the training compute, which means we could see much larger and more expensive systems if this growth continues.

Empirically derived scaling laws predict the performance of LLMs based on the given training budget, dataset size, and the number of parameters. This could mean that highly powerful systems will be concentrated in the hands of Big Tech.

The **KM scaling** law, proposed by Kaplan and colleagues, derived through empirical analysis and fitting of model performance with varied data sizes, model sizes, and training compute, presents power-law relationships, indicating a strong codependence between model performance and factors such as model size, dataset size, and training compute.

The **Chinchilla scaling law**, developed by the Google DeepMind team, involved experiments with a wider range of model sizes and data sizes and suggests an optimal allocation of compute budget to model size and data size, which can be determined by optimizing a specific loss function under a constraint.

However, future progress may depend more on data efficiency and model quality than sheer size. Though massive models grab headlines, computing power and energy constraints put a limit on unrestrained model growth. It's also unclear if performance will keep up further with the growth in parameters. The future could see the co-existence of massive, general models with smaller and more accessible specialized niche models that provide faster and cheaper training, maintenance, and inference.

It has already been shown that smaller specialized models can prove highly performant. As mentioned in *Chapter 6*, *Developing Software with Generative AI*, we've recently seen models such as phi-1 (*Textbooks Are All You Need*, 2023, Gunasekar and colleagues), with about 1 billion parameters, that – despite its smaller scale – achieve high accuracy on evaluation benchmarks. The authors suggest that improving data quality can dramatically change the shape of scaling laws.

Further, there is a body of work on simplified model architectures, which have substantially fewer parameters and only modestly drop accuracy (for example, *One Wide Feedforward is All You Need*, Pessoa Pires and others, 2023). Additionally, techniques such as fine-tuning, distillation, and prompting techniques can enable smaller models to leverage the capabilities of large foundations without replicating their costs. To compensate for model limitations, tools like search engines and calculators have been incorporated into agents, and multi-step reasoning strategies, plugins, and extensions may be increasingly used to expand capabilities.

The rapidly decreasing costs of AI model training represent a significant shift in the landscape, enabling broader participation in cutting-edge AI research and development. As noted, several factors are contributing to this trend, including optimization of training regimes, improvements in data quality, and the introduction of novel model architectures. Here is a brief summary of techniques and approaches for making generative AI more accessible and effective:

- **Simplified model architectures**: Streamlining model design for easier management, better interpretability, and lower computational cost.
- **Synthetic data generation**: Creating artificial training data to augment datasets while preserving privacy.
- **Model distillation**: Transferring knowledge from a large model into a smaller, more efficient one for easy deployment.
- **Optimized inference engines**: Software frameworks that increase the speed and efficiency of executing AI models on a given hardware.
- **Dedicated AI hardware accelerators**: Specialized hardware like GPUs and TPUs that dramatically accelerate AI computations.
- **Open-source and synthetic data**: High-quality public datasets enable collaboration and synthetic data enhances privacy and can help reduce bias.
- **Quantization**: Converting models to lower precision by reducing bit sizes of weights and activations, decreasing model size and compute costs.
- **Incorporating knowledge bases**: Grounding model outputs in factual databases reduces hallucinations and improves accuracy.
- **Retrieval augmented generation**: Enhancing text generation by retrieving relevant information from sources.
- **Federated learning**: Training models on decentralized data to improve privacy while benefiting from diverse sources.

Among the technical advancements helping drive down these costs, quantization techniques have emerged as an essential contributor. Open-source datasets and techniques such as synthetic data generation further democratize access to AI training by providing high-quality and data-efficient model development and removing some reliance on vast, proprietary datasets. Open-source initiatives contribute to the trend by providing cost-effective, collaborative platforms for innovation.

These innovations collectively lower barriers that have so far impeded real-world generative AI adoption across various segments:

- Financial barriers are reduced by compressing large model performance into far smaller form factors through quantization and distillation.
- Privacy risks are mitigated via federated and synthetic techniques circumventing exposure.
- The accuracy limitations hampering small models are relieved through grounding generation with external information.
- Specialized hardware exponentially accelerates throughput while optimized software maximizes the existing infrastructure.
- Democratizing access by tackling constraints like cost, security, and reliability unlocks benefits for vastly expanded audiences, steering generative creativity from a narrow concentration toward empowering diverse human talents.

The landscape is shifting from a focus on sheer model size and brute-force compute to clever, nuanced approaches that maximize computational efficiency and model efficacy. With quantization and related techniques lowering barriers, we're poised for a more diverse and dynamic era of AI development where resource wealth is not the only determinant of leadership in AI innovation.

This could mean a democratization of the market, as we'll see now.

Big Tech vs. small enterprises

As for the spread of technology, two primary scenarios exist. In the centralized scenario, generative AI and LLMs are primarily developed and controlled by large tech firms that invest heavily in the necessary computational hardware, data storage, and specialized AI/ML talent. Entities like these benefit from economies of scale and resources that allow them to bear the high costs of training and maintaining these sophisticated systems. They produce general models that are often made accessible to others through cloud services or APIs, but these one-size-fits-all solutions may not perfectly align with the requirements of every user or organization.

Conversely, in the self-service scenario, companies or individuals take on the task of training their own AI models. This approach allows for models that are customized to the specific needs and proprietary data of the user, providing more targeted and relevant functionality. However, this route traditionally requires significant AI expertise, substantial computational resources, and rigorous data privacy safeguards, which can be prohibitively expensive and complex for smaller entities.

The central question is how these scenarios will coexist and evolve. Presently, the centralized approach dominates due to the barriers in cost and expertise required for the self-service model. Yet with the democratization of AI – driven by declining computational costs, more widespread AI training and tools, and innovations that simplify model training – the self-service scenario may become increasingly viable for smaller organizations, local governments, and community groups. These groups could potentially harness tailored AI solutions for highly specific tasks, gaining advantages in agility and privacy preservation.

As these two business models continue to develop, a hybrid landscape may emerge where both approaches fulfill distinct roles based on use cases, resources, expertise, and privacy considerations. Large firms might continue to excel in providing industry-specific models, while smaller entities could increasingly train or fine-tune their own models to meet niche demands. The evolution of this landscape will largely depend on the pace of advancements that make AI more accessible, more cost-effective, and simpler to use without compromising robustness or privacy.

If robust tools emerge to simplify and automate AI development, custom generative models may even be viable for local governments, community groups, and individuals to address hyper-local challenges. While centralized Big Tech firms benefit currently from economies of scale, distributed innovation from smaller entities could unlock generative AI's full potential across all sectors of society.

While large tech firms currently dominate generative AI research and development, smaller entities may ultimately stand to gain the most from these technologies. As costs decline for computing, data storage, and AI talent, custom pre-training of specialized models could become feasible for small and mid-sized companies.

In a timeframe of 3–5 years, constraints around computing and talent availability could ease considerably, eroding the centralized moat created by massive investments. Specifically, if cloud computing costs decline as projected, and AI skills become more widespread through education and automated tools, self-training customized LLMs may become feasible for many companies.

Rather than relying on generic models from Big Tech, tailored generative AI fine-tuned on niche datasets could better serve unique needs. Start-ups and non-profits often excel at rapidly iterating to build cutting-edge solutions for specialized domains. Democratized access through cost reductions could enable such focused players to train performant models exceeding the capabilities of generalized systems.

In the next section, we'll discuss the potential of **Artificial General Intelligence (AGI)** and the threat of extinction by the malicious actions of a superintelligent artificial entity.

Artificial General Intelligence

Not all abilities in LLMs scale predictably with model size. Capabilities such as in-context learning may remain exclusive to particularly large models due to factors beyond raw computational growth. There's speculation that sustained scaling – training vast models on even larger datasets – might lead to broader skill sets and, some suggest, toward the development of AGI with reasoning abilities on par or beyond humans.

Nevertheless, current neuroscientific perspectives and the limitations of existing AI structures provide compelling arguments against an imminent leap to AGI (inspired by the discussion in the article *The feasibility of artificial consciousness through the lens of neuroscience* by Jaan Aru and others; 2023):

- **Lack of embodied, embedded information**: The current generation of LLMs lacks multimodal and embodied experiences, being trained predominantly on textual data. In contrast, human common sense and understanding of the physical world are developed through rich, diverse interactions involving multiple senses.
- **Different architecture from biological brains**: The relatively simple stacked transformer architecture used in models like GPT-4 lacks the complex recurrent and hierarchical structures of the thalamocortical system thought to enable consciousness and general reasoning in humans.
- **Narrow capabilities**: Existing models remain specialized for particular domains like text and fall short in flexibility, causal reasoning, planning, social skills, and general problem-solving intelligence. This could change either with increasing tool use or with fundamental changes to the models.
- **Minimal social abilities or intent**: Current AI systems have no innate motivations, social intelligence, or intent beyond their training objectives. Fears of malicious goals or desire for domination seem unfounded.
- **Limited real-world knowledge**: Despite ingesting huge datasets, the factual knowledge and common sense of large models remain very restricted compared to humans. This impedes applicability in the physical world.
- **Data-driven limitations**: Reliance on pattern recognition from training data rather than structured knowledge makes reliable generalization to novel situations difficult.

As we address pressing AI challenges, the discourse around AI's threat and its potential for societal disruption should not overshadow immediate issues like fairness and privacy.

Given current model limitations and the lack of agency, the notion of today's AI rapidly evolving into a dangerous superintelligence appears highly unlikely. In formulating regulations, we must be vigilant against regulatory capture, where dominant industry players invoke far-fetched scenarios of AI-driven destruction to distract from pressing concerns and to shape rules to fit their interests, potentially marginalizing the concerns of smaller entities and the public. Nonetheless, ongoing attention to safety research and ethical concerns is essential, especially as AI advances.

Let's discuss the broader economy, and – the elephant in the room – jobs!

Economic consequences

Integrating generative AI promises immense productivity gains through automating tasks across sectors – albeit risking workforce disruptions given the pace of change. Assuming computing scales sustainably, projections estimate 30–50% of current work activities will be automatible by 2030, adding $6–8 trillion annually to global GDP. Sectors like customer service, marketing, software engineering, and R&D may see over 75% of use case value. However, past innovations ultimately spawned new occupations, suggesting long-term realignment.

Developed regions are likely to witness faster uptake, displacing administrative, creative, and analytical roles initially. Yet automation extends beyond employment loss – at present, under 20% of US worker tasks seem automatable directly through LLMs. But LLM-enhanced software could transform 50% of tasks, affirming the force multiplication from complementary innovations.

Thus automation's labor impact remains complex – while augmenting productivity, transitional pains persist. Still, the virtuous cycle between AI progress and emerging specializations signals hopes for an uplift over redundancy. And braiding priorities of sustainability, equity, and human dignity throughout this transformation promises optimizing empowerment over exploitation.

In a professional context, generative AI is poised to amplify human creativity and transform traditional workflows across a range of industries. For content creators, such as marketers and journalists, AI can rapidly generate initial drafts, fostering a baseline that human creativity can build upon for more customized outputs. Software developers benefit from AI's ability to produce code snippets, helping to expedite the development process. For scholars and scientists, the ability of AI to distill complex research into comprehensive summaries can catalyze scholarly progress and innovation.

Here are some key predictions about how jobs may be impacted by advances in language models and generative AI:

- Routine legal work like draft preparation will be increasingly automated, changing job roles for junior lawyers and paralegals.
- Software engineering will see a rise in AI coding assistants handling mundane tasks, enabling developers to focus on complex problem-solving.
- Data scientists will spend more time refining AI systems rather than building predictive models from scratch.
- Demand for specialized roles like prompt engineering will continue to rise.
- Teachers will utilize AI for course preparation and personalized student support.
- Journalists, paralegals, and graphic designers will employ generative AI to enhance content creation, raising concerns about job impacts.
- Demand will grow for experts in AI ethics, regulations, and security to oversee responsible development.
- Musicians and artists will collaborate with AI, boosting creative expression and accessibility.
- Striking an optimal balance between AI capabilities and human judgment will be vital across sectors.
- The common thread is that while routine tasks face increasing automation, human expertise to steer AI directions and ensure responsible outcomes will remain indispensable.

While certain jobs may be displaced by AI in the near term, especially routine cognitive tasks, it may automate certain activities rather than eliminate entire occupations. Technical experts like data scientists and programmers will remain key to developing AI tools and realizing their full business potential. By automating rote tasks, models may free up human time for higher-value work, boosting economic output.

Concerns have emerged about saturation as generative AI tools are relatively easy to build using foundation models. Customization of models and tools will allow value creation, but it's unclear who will capture the most upsides and how powerful these applications can be. While current market hype is high, investors are tempering decisions given lower valuations and skepticism following the 2021 AI boom/bust cycle. The long-term market impact and the winning generative AI business models have yet to unfold.

The **2021 AI boom/bust cycle** refers to a rapid acceleration in investment and growth in the AI start-up space followed by a market cooldown and stabilization in 2022 as projections failed to materialize and valuations declined.

Here's a quick summary:

- **Boom phase (2020-2021)**: There was huge interest and skyrocketing investment in AI start-ups offering innovative capabilities like computer vision, natural language processing, robotics, and machine learning platforms. Total funding for AI start-ups hit record levels in 2021, with over $73 billion invested globally according to Pitchbook. Hundreds of AI start-ups were founded and funded during this period.
- **Bust phase (2022)**: In 2022, the market underwent a correction, with valuations of AI start-ups falling significantly from their 2021 highs. Several high-profile AI start-ups like Anthropic and Cohere faced valuation markdowns. Many investors became more cautious and selective with funding AI start-ups. Market corrections in the broader tech sector also contributed to the bust.
- **Key factors**: Excessive hype, unrealistic growth projections, historically high valuations in 2021, and broader economic conditions all contributed to the boom-bust cycle. The cycle followed a classic pattern seen previously in sectors like dot-com and blockchain.

As AI models become more sophisticated and economical to operate, we can anticipate a substantial proliferation of generative AI and LLM applications into novel domains. Beyond just the plummeting hardware expenses that have historically followed Moore's Law, there are additional economies of scale affecting AI systems.

In *Chapter 1*, *What Is Generative AI?*, we discussed the pertinent trend in the AI industry that encompasses gains in efficiency stemming from the iterative refinement of code, the development of sophisticated tools, and the enhancement of techniques. The improved efficiency because of new techniques and approaches, combined with the declining hardware costs, fosters a virtuous cycle: as costs diminish, AI adoption widens, in turn spurring further cost reductions and efficiency improvements. What emerges is a feedback loop where each iteration of efficiency catalyzes increased usage, which in itself leads to even greater efficiency – a dynamic poised to dramatically advance the frontier of AI capabilities.

Let's look at various sectors where generative models will have profound near-term impacts, starting with creative endeavors.

Creative industries and advertising

The gaming and entertainment industries are leveraging generative AI to craft uniquely immersive user experiences. Major efficiency gains from automating creative tasks could increase leisure time spent online. Generative AI can enable machines to generate new and original content, such as art, music, and literature, by learning from patterns and examples. This has implications for creative industries, as it can enhance the creative process and potentially create new revenue streams. It also unlocks new scales of personalized, dynamic content creation for media, film, and advertising.

For media, film, and advertising, AI unlocks new scales of personalized, dynamic content creation. In journalism, automated article generation using massive datasets can free up reporters to focus on more complex investigative stories. **AI-Generated Content** (**AIGC**) is playing a growing role in transforming media production and delivery by enhancing efficiency and diversity. In journalism, text generation tools automate writing tasks traditionally done by human reporters, significantly boosting productivity while maintaining timeliness. Media outlets like the Associated Press generate thousands of stories per year using AIGC. Robot reporters like the Los Angeles Times Quakebot can swiftly produce articles on breaking news.

Other applications include Bloomberg News' Bulletin service where chatbots create personalized one-sentence news summaries. AIGC also enables AI news anchors that co-present broadcasts with real anchors by mimicking human appearance and speech from text input. Chinese news agency Xinhua's virtual presenter Xin Xiaowei is an example, presenting broadcasts from different angles for an immersive effect.

AIGC is transforming movie creation from screenwriting to post-production. AI screenwriting tools analyze data to generate optimized scripts. Visual effects teams blend AI-enhanced digital environments and de-aging with live footage for immersive visuals. Deep fake technology recreates or revives characters convincingly.

AI also powers automated subtitle generation, even predicting dialogue in silent films by training models on extensive audio samples. This expands accessibility via subtitles and recreates voice-overs synchronized to scenes. In post-production, AI color grading and editing tools like Colourlab AI and Descript simplify processes like color correction using algorithms.

In advertising, AIGC unlocks new potential for efficient, customized advertising creativity and personalization. AI-generated content allows advertisers to create personalized, engaging ads tailored to individual consumers at scale. Platforms like **Creative Advertising System** (**CAS**) and **Smart Generation System Personalized Advertising Copy** (**SGS-PAC**) leverage data to automatically generate ads with messaging targeted to specific user needs and interests.

AI also assists in advertising creativity and design – tools like Vinci produce customized attractive posters from product images and slogans, while companies like Brandmark.io generate logo variations based on user preferences. GAN technologies automate product listing generation with keywords for effective peer-to-peer marketing. Synthetic ad production is also on the rise, enabling highly personalized, scalable campaigns that save time.

In music, tools like Google's Magenta, IBM's Watson Beat, and Sony CSL's Flow Machine can generate original melodies and compositions. AIVA similarly creates unique compositions from parameters tuned by users. LANDR's AI mastering uses machine learning to process and improve digital audio quality for musicians.

In visual arts, MidJourney uses neural networks to generate inspirational images that can kickstart painting projects. Artists have used its outputs to create prize-winning works. DeepDream's algorithm imposes patterns on images, creating psychedelic art. GANs can generate abstract paintings converging on a desired style. AI painting conservation analyzes artwork to digitally repair damage and restore pieces.

Animation tools like Adobe's Character Animator and Anthropic's Claude can help with the generation of customized characters, scenes, and motion sequences, opening animation potential for non-professionals. ControlNet adds constraints to steer diffusion models, increasing output variability.

For all these applications, advanced AI expands creative possibilities through both generative content and data-driven insights. In all cases, quality control and properly attributing the contributions of human artists, developers, and training data remains an ongoing challenge as adoption spreads.

Education

One potential near-future scenario is that the rise of personalized AI tutors and mentors could democratize access to education for high-demand skills aligned with an AI-driven economy. In the education sector, generative AI is already transforming how we teach and learn. Tools like ChatGPT can be used to automatically generate personalized lessons and customized content for individual students. This reduces instructor workloads substantially by automating repetitive teaching tasks. AI tutors provide real-time feedback on student writing assignments, freeing up teachers to focus on more complex skills. Virtual simulations powered by generative AI can also create engaging, tailored learning experiences adapted to different learners' needs and interests.

However, risks around perpetuating biases and spreading misinformation need to be studied further as these technologies evolve. The accelerating pace of knowledge and the obsolescence of scientific findings mean that training children's curiosity-driven learning should focus on developing the cognitive mechanisms involved in initiating and sustaining curiosity, such as awareness of knowledge gaps and the use of appropriate strategies to resolve them.

While AI tutors tailored to each student could enhance outcomes and engagement, poorer schools may be left behind, worsening inequality. Governments should promote equal access to prevent generative AI from becoming a privilege of the affluent. Democratizing opportunity for all students remains vital.

If implemented thoughtfully, personalized AI-powered education could make crucial skills acquisition accessible to anyone motivated to learn. Interactive AI assistants that adapt courses to students' strengths, needs, and interests could make learning efficient, engaging, and equitable. However, challenges around access, biases, and socialization need addressing.

Law

Generative models like LLMs can automate routine legal tasks such as contract review, documentation generation, and brief preparation. They also enable faster, comprehensive legal research and analysis. Additional applications include explaining complex legal concepts in plain language and predicting litigation outcomes using case data. However, responsible and ethical use remains critical given considerations around transparency, fairness, and accountability. Overall, properly implemented AI tools promise to boost legal productivity and access to justice while requiring ongoing scrutiny regarding reliability and ethics.

Manufacturing

In the automotive sector, generative models are employed to generate 3D environments for simulations and aid in the development of cars. Additionally, generative AI is utilized for road-testing autonomous vehicles using synthetic data. These models can also process object information to comprehend the surrounding environment, understand human intent through dialogues, generate natural language responses to human input, and create manipulation plans to assist humans in various tasks.

Medicine

A model that can accurately predict physical properties from gene sequences would represent a major breakthrough in medicine and could have profound impacts on society. It could further accelerate drug discovery and precision medicine, enable earlier disease prediction and prevention, provide a deeper understanding of complex diseases, and improve gene therapies. However, it also raises major ethical concerns around genetic engineering and could exacerbate social inequalities.

New techniques with neural networks are already employed to lower long-read DNA sequencing error rates (Baid and colleagues; *DeepConsensus improves the accuracy of sequences with a gap-aware sequence transformer*, September 2022), and, according to a report by ARK Investment Management (2023), in the short term, technology like this can make it already possible to deliver the first high-quality, whole long-read genome for less than $1,000. This means that large-scale gene-to-expression models might not be far away either.

Military

Militaries worldwide are investing in research to develop **Lethal Autonomous Weapons Systems (LAWS)**. Robots and drones can identify targets and deploy lethal force without any human supervision. Machines can process information and react faster than humans, removing emotion from lethal decisions. However, this raises significant moral questions. Allowing machines to determine whether lives should be taken crosses a troubling threshold. Even with sophisticated AI, complex factors in war like proportionality and distinction between civilians and combatants require human judgment.

If deployed, completely autonomous lethal weapons would represent an alarming step toward relinquishing control over life-and-death decisions. They could violate international humanitarian law or be used by despotic regimes to terrorize populations. Once unleashed fully independently, the actions of autonomous killer robots would be impossible to predict or restrain.

The advent of highly capable generative AI will likely transform many aspects of society in the coming years beyond the economics and the disruption of certain jobs. Let's think a bit more broadly about the societal impact!

Societal implications

As generative models continue to develop and add value to businesses and creative projects, generative AI will shape the future of technology and human interaction across domains. While their widespread adoption brings forth numerous benefits and opportunities for businesses and individuals, it is crucial to address the ethical and societal concerns that arise from increasing reliance on AI models in various fields.

Generative AI offers immense potential benefits across personal, societal, and industrial realms if deployed thoughtfully. At a personal level, these models can enhance creativity and productivity, and increase accessibility to services like healthcare, education, and finance. By democratizing access to knowledge resources, they can help students learn or aid professionals in making decisions by synthesizing expertise. As virtual assistants, they provide instant, customized information to facilitate routine tasks.

From a consumer standpoint, generative AI has the potential to deliver unprecedented personalization. Recommendation systems can fine-tune their outputs to individual preferences. Marketing efforts can be adapted to specific customer segments and local tastes while maintaining consistency and scale.

The rise of generative AI represents a significant milestone within a broader societal trend of how creative content is being generated and consumed. The internet has already nurtured a culture of remixing, where derivative works and co-creation are the norms. Generative AI fits naturally within this paradigm by creating new content through the recombination of existing digital materials, promoting the ethos of shared, iterative creation.

However, the capacity of generative AI to synthesize and remix copyrighted materials at scale presents intricate legal and ethical challenges. The training of these models on extensive corpora that encompass literature, articles, images, and other copyrighted works creates a tangled web for attribution and compensation. Existing tools struggle to identify content generated by AI, which complicates efforts to apply traditional copyright and authorship principles. This dilemma underscores the urgent need for legal frameworks that can keep pace with technological advances and navigate the complex interplay between rights-holders and AI-generated content.

One of the major problems that I can see is misinformation, either in the interest of political interest groups, foreign actors, or large corporations. Let's discuss this threat!

Misinformation and cybersecurity

AI presents a dual-edged sword against disinformation. While it enables scalable detection, automation makes it easier to spread sophisticated, personalized propaganda. AI could help or harm security depending on whether it is used responsibly. It increases vulnerabilities to misinformation along with cyberattacks using generative hacking and social engineering.

There are significant threats associated with AI techniques like micro-targeting and deepfakes. Powerful AI can profile users psychologically to deliver personalized disinformation that facilitates concealed manipulation, escaping broad examination. Big Data and AI could be leveraged to exploit psychological vulnerabilities and infiltrate online forums to attack and spread conspiracy theories.

Disinformation has transformed into a multifaceted phenomenon, involving biased information, manipulation, propaganda, and intent to influence political behavior. For example, during the COVID-19 pandemic, the spread of misinformation and infodemics has been a major challenge. AI can influence public opinion and sway elections.

It can also generate fake audio/video content to damage reputations and sow confusion. State and non-state actors are weaponizing these capabilities for propaganda to damage reputations and sow confusion. AI can be used by political parties, governments, criminal groups, and even the legal system to launch lawsuits and/or extract money.

This likely will have far-reaching consequences in various domains. A significant portion of internet users may be obtaining the information they need without accessing external websites. There is a danger of large corporations being the gatekeepers of information and controlling public opinion, effectively being able to restrict certain actions or viewpoints.

Careful governance and digital literacy are essential to build resilience. Though no single fix exists, collective efforts promoting responsible AI development can help democratic societies address emerging threats.

Let's talk more about regulations!

Regulations and implementation challenges

Realizing the potential of generative AI in a responsible manner involves addressing a number of practical legal, ethical, and regulatory issues:

- **Legal**: Copyright laws remain ambiguous regarding AI-generated content. Who owns the output – the model creator, training data contributors, or end users? Replicating copyrighted data in training also raises fair use debates that need clarification.
- **Data protection**: Collecting, processing, and storing the massive datasets required to train advanced models creates data privacy and security risks. Governance models ensuring consent, anonymity, and safe access are vital.
- **Oversight and regulations**: Calls are mounting for oversight to ensure non-discrimination, accuracy, and accountability from advanced AI systems. However, flexible policies balancing innovation and risk are needed rather than burdensome bureaucracy.
- **Ethics**: Frameworks guiding development toward beneficial outcomes are indispensable. Integrating ethics through design practices focused on transparency, explicability, and human oversight helps build trust.

Overall, proactive collaboration between policymakers, researchers, and civil society is essential to settle unresolved issues around rights, ethics, and governance. With pragmatic guardrails in place, generative models can fulfill their promise while mitigating harm.

There is a growing demand for algorithmic transparency. This means that tech companies and developers should reveal the source code and inner workings of their systems. However, there is resistance from these companies and developers, who argue that disclosing proprietary information would harm their competitive advantage. Open-source models will continue to thrive, and local legislation in the EU and other countries will push for transparent use of AI.

The consequence of AI bias includes potential harm to individuals or groups due to biased decisions made by AI systems. Incorporating ethics training into computer science curricula can help reduce biases in AI code. By teaching developers how to build applications that are ethical by design, the probability of biases being embedded into the code can be minimized. To stay on the right path, organizations need to prioritize transparency, accountability, and guardrails to prevent bias in their AI systems. AI bias prevention is a long-term priority for many organizations; however, without legislation driving it, it can take time to be introduced. Local legislation in EU countries, for example, such as the European Commission's proposal for harmonized rules on AI regulation, will drive more ethical use of language and imagery.

A current German law on fake news, which imposes a 24-hour timeframe for platforms to remove fake news and hate speech, is impractical for both large and small platforms. Additionally, the limited resources of smaller platforms make it unrealistic for them to police all content. Further, online platforms should not have the sole authority to determine what is considered truth, as this could lead to excessive censorship. More nuanced policies are needed that balance free speech, accountability, and feasibility for a diversity of technology platforms to comply. Relying solely on private companies to regulate online content raises concerns about a lack of oversight and due process. Broader collaboration between government, civil society, academics, and industry can develop more effective frameworks to counter misinformation while protecting rights.

To maximize benefits, companies need to ensure human oversight, diversity, and transparency in development. Policymakers may need to implement guardrails preventing misuse while providing workers with support to transition as activities shift. With responsible implementation, generative AI could propel growth, creativity, and accessibility in a more prosperous society. Addressing potential risks early on and ensuring a just distribution of benefits designed to serve public welfare will cultivate a sense of trust among stakeholders, such as:

- **The dynamics of progress**: Fine-tuning the pace of transformation is critical to avoid any undesired repercussions. Moreover, excessively slow developments could stifle innovation, suggesting that determining an ideal pace through encompassing public discourse is crucial.
- **The human-AI symbiosis**: Rather than striving for outright automation, more advantageous systems would integrate and complement the creative prowess of humans with the productive efficiency of AI. Such a hybrid model will ensure optimal oversight.
- **Promoting access and inclusion**: Equitable access to resources, relevant education, and myriad opportunities concerning AI is key to negating the amplification of disparities. Representativeness and diversity should be prioritized.
- **Preventive measures and risk management**: Constant evaluation of freshly emerging capabilities via interdisciplinary insights is necessary to evade future dangers. Excessive apprehensions, however, should not impede potential progress.
- **Upholding democratic norms**: Collaborative discussions, communal efforts, and reaching a compromise will inevitably prove more constructive in defining the future course of AI, as compared to unilateral decrees imposed by a solitary entity. Public interest must take precedence.

Let's conclude this chapter!

The road ahead

The forthcoming era of generative AI models offers a plethora of intriguing opportunities and unparalleled progression, yet it is interspersed with numerous uncertainties. As discussed in this book, many breakthroughs have been accomplished in recent months, but successive challenges continue to linger, mainly pertaining to precision, reasoning ability, controllability, and entrenched bias within these models. While grandiose claims of superintelligent AI on the horizon may seem hyperbolic, consistent trends predict sophisticated capabilities sprouting within a few decades.

On a technical level, generative models like ChatGPT often function as black boxes, with limited transparency into their decision-making processes. A lack of model interpretability makes it difficult to fully understand model behavior or to control outputs. There are also concerns about potential biases that could emerge from imperfect training data. On a practical level, generative models require extensive computational resources for training and deployment; however, we discussed developments and trends that change that.

On the positive side, AI can democratize skills, allowing amateurs to produce professional quality output in design, writing, and other areas. Businesses can benefit from faster, cheaper, on-demand work. However, there are major concerns about job losses, especially for specialized middle-class roles like graphic designers, lawyers, and doctors. Their work is being automated while low-skilled workers learn to leverage AI as a superpower.

However, the proliferation of generative content raises valid concerns about misinformation, plagiarism in academia, and impersonation in online spaces. As these models become more adept at mimicking human expression, people may have difficulty discerning what is human-generated versus AI-generated, enabling new forms of deception. Deepfakes produced in real-time will proliferate scams and erode trust. Most ominously, AI could be weaponized by militaries, terrorists, criminals, and governments for propaganda and influence. There are also fears about generative models exacerbating social media addiction due to their ability to produce endless customized content.

The sheer pace of advancement creates unease surrounding human obsolescence and job displacement, which could further divide economic classes. Unlike physical automation of the past, generative AI threatens cognitive job categories previously considered safe from automation. Managing this workforce transition ethically and equitably will require foresight and planning. There are also philosophical debates around whether AI should be creating art, literature, or music that has historically reflected the human condition.

For corporations, effective governance frameworks have yet to be established around acceptable use cases. Generative models amplify risks of misuse, ranging from creating misinformation such as deepfakes to generating unsafe medical advice. Legal questions around content licensing and intellectual property arise. While generative models can enhance business productivity, quality control and bias mitigation incur costs.

Looking decades ahead, perhaps the deepest challenges are ethical. As AI is entrusted with more consequential decisions, alignment with human values becomes critical. While accuracy, reasoning ability, controllability, and mitigating bias remain technical priorities, other priorities should include fortifying model robustness, promoting transparency, and ensuring alignment with human values.

While future capabilities remain uncertain, proactive governance and democratization of access are essential to direct these technologies toward equitable, benevolent outcomes. Collaboration between researchers, policymakers, and civil society around issues of transparency, accountability, and ethics can help align emerging innovations with shared human values. The goal should be to empower human potential, not mere technological advancement.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

`https://packt.link/lang`