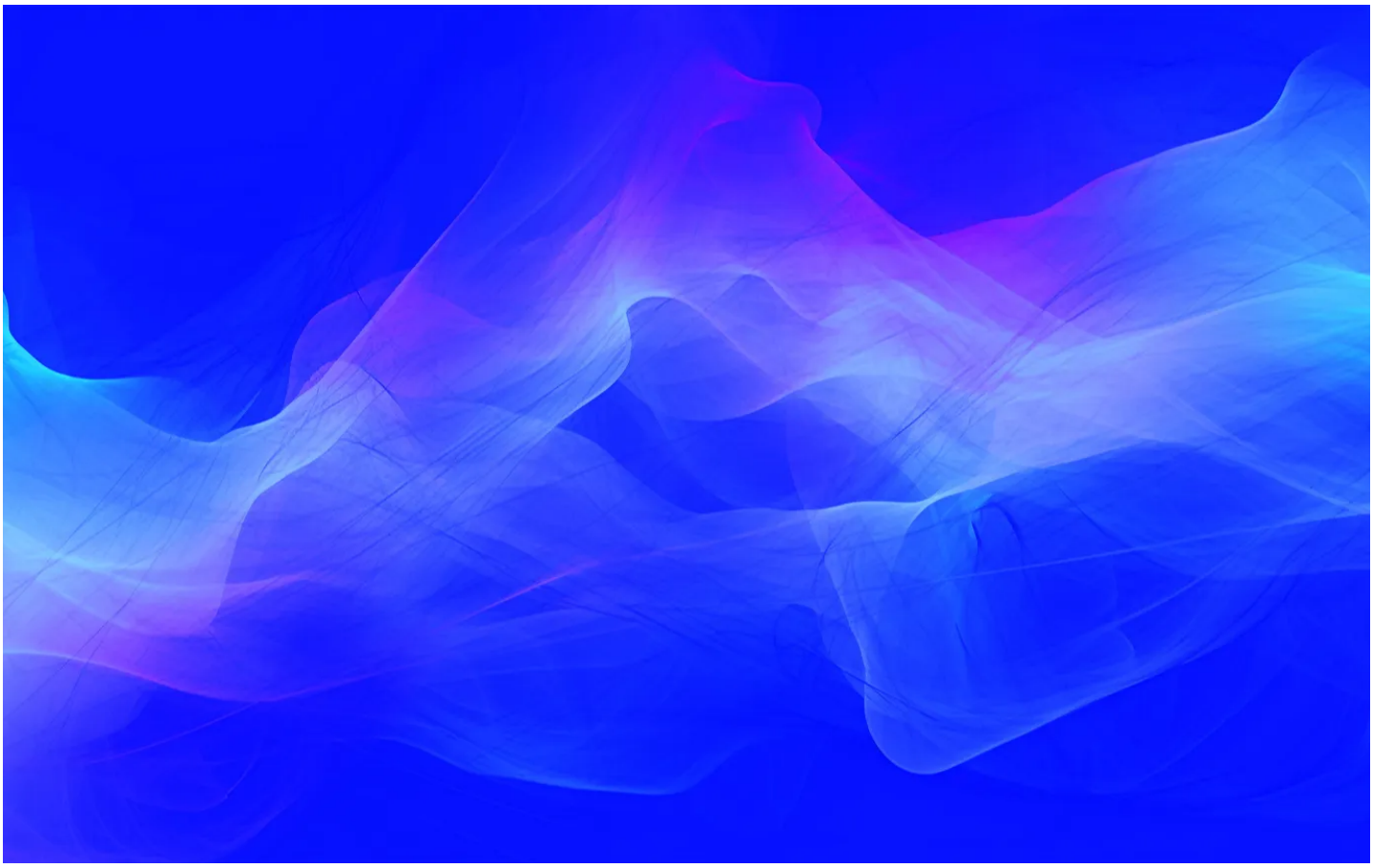
BOREALIS AI

Blog

A High-level Overview of Large Language Models

Jul. 12, 2023

W. Zi, L. El Asri, S. Prince



Since 2022, a series of AI systems have been introduced that enable machines to read, analyze, interpret, and derive meaning from human language. One such system is ChatGPT, which gained a over a hundred million users within a mere two months of its launch in November 2022. Its successor, GPT-4 was released in March 20 performance in many tasks, including passing the Ba

Figure 1. Machine learning model performance over time on the MMLU benchmark. The MMLU covers 57 tasks such as basic mathematics, U.S. history, computer science, legal matters, etc., to evaluate a model's language understanding ability. Over time, the state-of-the-art models have shown remarkable improvements in their scores. In early 2019, GPT-2 achieved a score of 32.4, and in early 2023, GPT-4 achieved a score of 86.4.

These remarkable advances are powered by *large language models* (LLMs), which have improved AI's language understanding capabilities by nearly threefold since 2019 (Figure 1). However, despite their impressive capabilities, LLMs still have limitations. For example, users have observed instances where ChatGPT cites papers that don't exist. Both the remarkable achievements and the ongoing challenges highlight the importance of understanding LLMs better. This will allow us to use them to create practical applications and will help us improve their accessibility and ethical behaviour.

This blog presents a high-level introduction to key concepts for understanding LLMs and answers questions including:

- How do we train and fine-tune a large language model?
- What is prompt engineering, and how does it work?
- Why are people concerned about large language models?

This introduction is suited for those with a basic understanding of machine learning who wish to familiarize themselves with LLMs. We avoid complex mathematical formulas and instead employ easy-to-understand illustrations and examples. For those seeking a deeper dive, we provide a list of recommended reading materials at the end.

What is a large language model?

A *language model* is built to process and understand a text input *(prompt)*, and then generate a text output (response) accordingly. These models are
unlabeled text, allowing them to learn general linguis
base. The primary distinction between a regular lang
lies in the number of parameters used. While there is n

Most LLMs, including all models from the GPT family, are *auto-regressive* language models. They (i) predict the next word based on the preceding text provided as the input, (ii) append the predicted word back to the input, and (iii) repeat this process as needed. This allows them to generate extensive bodies of text (Figure 2). This process involves three key technical components. First, the input is represented using *token embeddings*. Second, the probability of the next word is predicted using a *decoder-only Transformer*, and finally, the next word is selected based on probabilities. These stages are discussed in the following three sections.

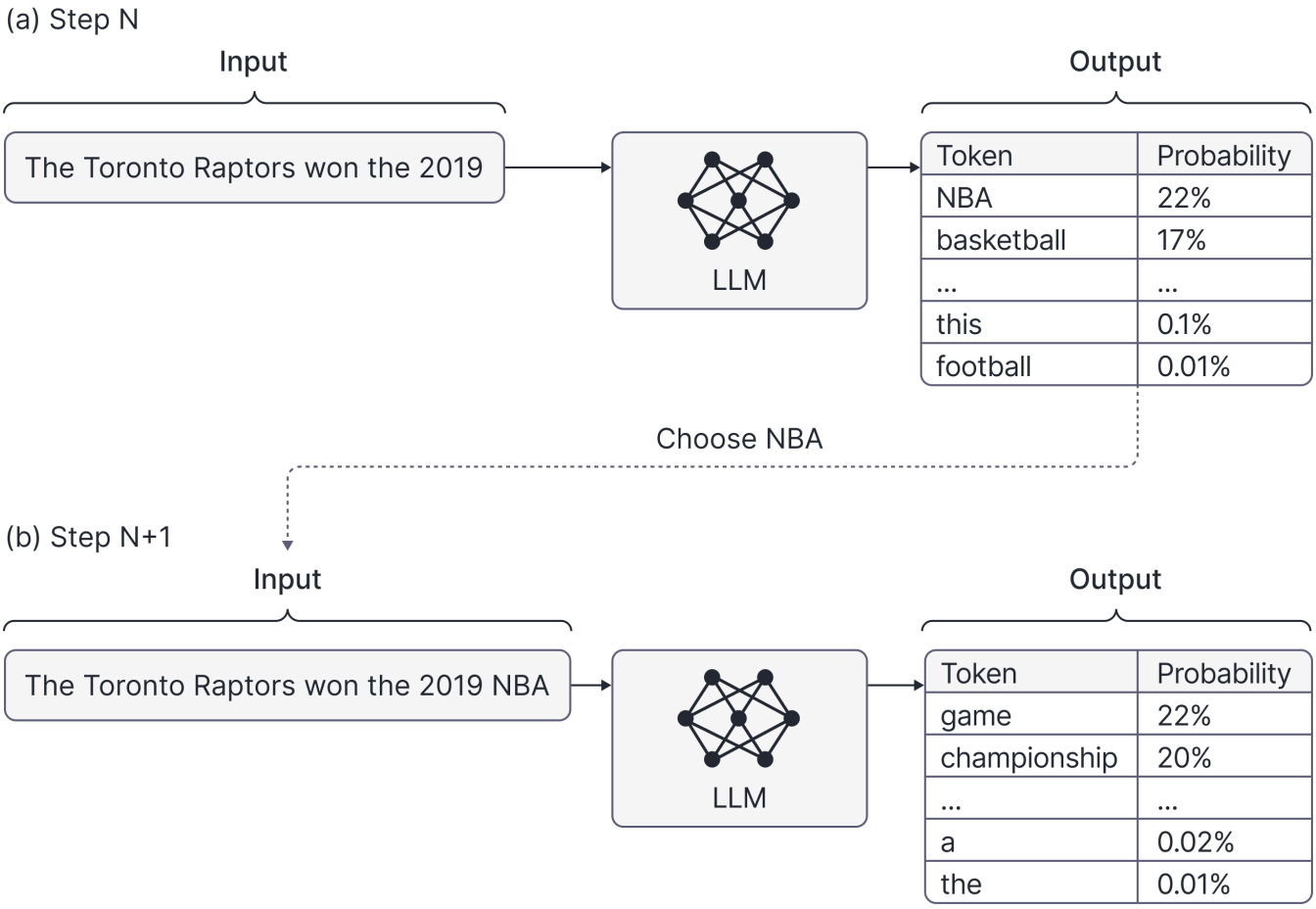


Figure 2. Autoregressive language model. (a) At step N, after receiving the input "The Toronto Raptors won the 2019", the LLM generates a probability distribution over the entire vocabulary and chooses the subsequent word based on the distribution (e.g., the most probable word, "NBA"). (b) At step N+1, the chosen subsequent word "NBA" is appended to the original input and forms the updated input "The Toronto Raptors won the 2019 NBA". The LLM then generates the probability distribution over the entire vocabulary and identifies "game" as the most probable word for completion.

Input representation

When the LLM receives a text input, a tokenizer divides it into a sequence of *tokens*, each of which represents a word or part of a word. Each token i
known as a token embedding (Figure 3). These toker
using rule-based methods (e.g., tf-idf) or learned thr
expected to contain the syntactic and semantic meanin

delineate the start and end of sections of input text).

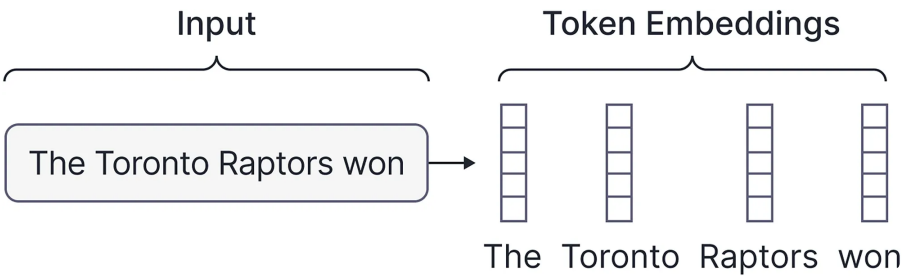


Figure 3. Input representation. The input "The Toronto Raptors won" is tokenized to a sequence of tokens: "The", "Toronto", "Raptors" and "won". Each token is then mapped to its own unique token embeddings with the same fixed length.

Each language model has a *context window* length, which determines the maximum number of tokens it can process at a time. Typically, LLMs have a context window of a few thousand tokens.

Decoder-only Transformer

Various LLM model architectures have been experimented with (Yang et al., 2023). However, the *decoder-only Transformer*, a modified version of the original Transformer (Vaswani et al., 2017), has emerged as the preferred choice.

The decoder-only Transformer consists of a stack of Transformer *decoder blocks* (Figure 4). Each decoder block comprises a *self-attention* layer and a *feedforward network* (Figure 5). The self-attention layer enables tokens to interact with one another, while the feedforward network independently applies the same transformations to each token embedding without token-level interactions. Additional components, such as layer normalization and residual connections, are included to facilitate training.

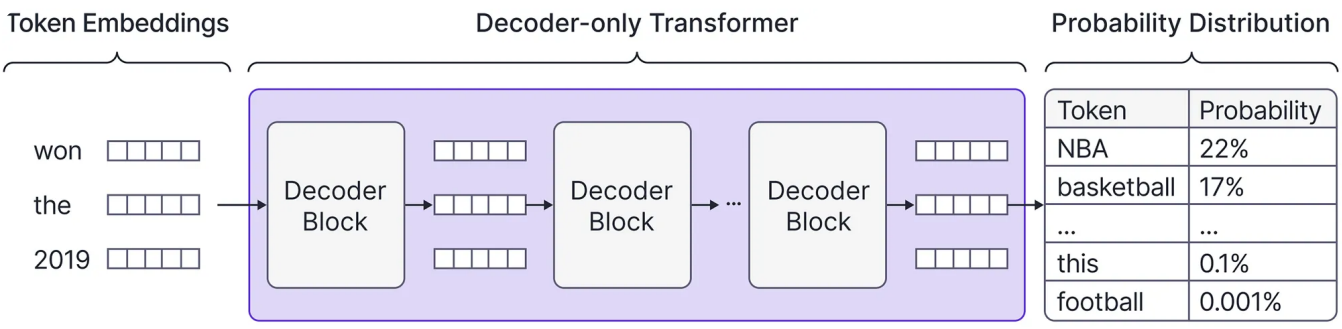


Figure 4. Transformer architecture. A Decoder-only Transformer con
the input is converted to a sequence of token embeddings (Figure
transforms the input embeddings block by block. At the end, a prol
produced.

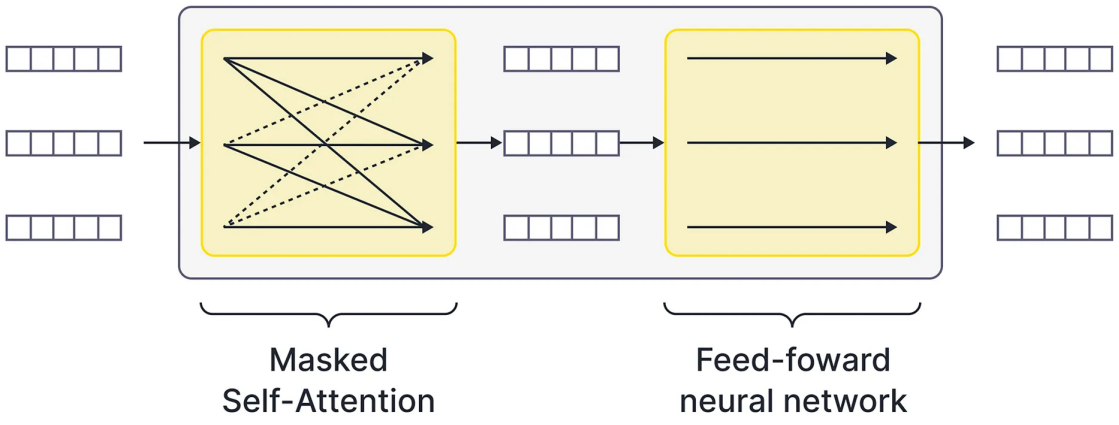


Figure 5. Transformer decoder block. A Transformer decoder block contains two crucial components: a masked self-attention layer and a feed-forward neural network. The masked self-attention layer enables pairwise interactions between the tokens (arrows). The dashed lines indicate where interaction between the tokens is disabled; masked self-attention only considers input embeddings from tokens before the prediction target, so all upward-facing arrows are dashed (disabled). Subsequently, the same feed-forward network is applied to each embedding. This acts as a nonlinear transformation function to modify the input embeddings independently.

Contents

of) all I input embeddings. The weight used to combine the i^{th} input embedding depends on the similarity between (a processed version of) the k^{th} input embedding and (a processed version of) the i^{th} input embedding.

In practice, the weights used to compute the k^{th} output embedding are set to zero for input embeddings that are further ahead in the sequence. This is known as *masked self-attention* (Figure 5) and makes training more efficient; during training, the model is now required to predict the next token for *every* input sequence simultaneously, and the masking means that it cannot cheat by looking ahead.

The self-attention mechanism allows the embedding to capture a mixture of information from the entire preceding sequence, focusing on important parts and diminishing the influence of the irrelevant parts. For a detailed explanation, see our previous blog and Alammar (2018).

Next word selection

At each step, an LLM produces a probability distribution over the entire vocabulary. This distribution is used to choose the next token to continue the sequence. There are two obvious approaches: (i) selecting the most likely token or (ii) randomly drawing a token according to the probabilities. However, each approach has drawbacks. The first approach often leads to generic text, while the second approach can occasionally result in implausible continuations when a token with low probability (of which there are many) is c
retain only the k most probable tokens based on rank
probability mass adding up to a certain threshold *(top*

Scaling laws

Numerous studies (e.g., Kaplan et al. 2020) have revealed that increasing the size of language models can result in improved performance. This insight has motivated researchers to explore the capabilities of machine learning systems in tackling complex tasks by employing larger models. Common approaches include stacking together more Transformer decoder blocks, increasing the window length, and aggregating results from parallel self-attention layers (so-called *multi-head attention*).

Consequently, the number of parameters used by LLMs has grown over time. For instance, GPT-2 (Radford et al., 2019), the largest language model available before 2019, had only 1.5 billion parameters. In contrast, Google's PaLM (Chowdhery et al., 2022), boasted a staggering 540 billion parameters, which is approximately 360 times more than GPT-2 (Figure 6). At the time of writing, the largest known model has an astounding 1.2 trillion parameters Du et al. (2022).

How to train an LLM

This section outlines the three methods for training LLMs (Figure 7): *pre-training*, *instruction fine-tuning*, and *reinforcement learning from human feedback* (RLHF). Each method plays a distinct role in the training process and contributes to the success of LLMs. It's important to note that while pre-training is mandatory, the other methods (which are applied after pre-training) are optional.

Pre-training

To ensure LLMs generate coherent and meaningful text, they need to be equipped with a comprehensive understanding of language and the world. *Pre-training* involves leveraging unlabeled text to learn a universal language represen⁺ follows that pre-training requires a substantial amou on 780 billion tokens. And by following the recipe of L

pages. Once this data is collected, pre-training optimizes the model's parameters to maximize the probability of correctly predicting the next token.

Pre-training serves is a crucial initial step in the training process. It offers significant performance improvements for downstream tasks (Radford et al., 2018) relative to training on these tasks alone. Brown et al. (2020) also revealed the remarkable *few-shot learning* capabilities of large pre-trained language models. By providing just a few examples as input, these models can perform new tasks they have never encountered before (Figure 8).

80 GB of RAM each. The estimated cost of this training process exceeds four million dollars, making it an exceptionally expensive undertaking.

Instruction fine-tuning

Pretrained LLMs, such as GPT-3, have demonstrated remarkable performance in various language tasks. However, there are still areas where the performance of these models is lacking. For example, their few-shot learning capabilities face challenges when it comes to tasks like determining the truthfulness of a hypothesis based on some input content. Also, they are limited in their ability to perform in zero-shot learning, where the model is expected to generate responses solely based on instructions without any examples provided (Brown et al., 2020).

To enhance the model's ability to follow instructions and generate responses aligned with the ground truth, *instruction fine-tuning*, or *instruction tuning* for short, is introduced. This involves fine-tuning a pre-trained LLM using labeled demonstration data. Some researchers also refer to this process as *supervised fine-tuning* (e.g., Ouyang et al., 2022).

There are three main methods for collecting the demonstration data required for instruction fine-tuning. The first method leverages existing labeled datasets created for language-related tasks (Figure 9a). For example, Wei et al. (2021) use a set of predefined templates to convert labeled datasets (Raffel et al., 2020) into text-based prompts and responses. The process transforms labels from classification tasks into class names and represents numbers from regression tasks as plain text. By unifying all datasets created for language-related tasks into a text-to-text representation, the model can be trained using a consistent framework.

A second method for collecting demonstration data is to
prepare it (Figure 9b). For training InstructGPT, Open
GPT-3 API submissions and recruited 40 labelers to
2022). This allows the model to learn from real questio

The third method, known as "self-instruct" (Wang et al., 2023) (Figure 9c) involves using the pre-trained LLM to generate its own training data. In this approach, individuals only need to prepare one prompt per task for demonstration. The pre-trained LLM is then used to generate a series of prompts similar to the example prompts and the corresponding responses to these generated prompts.

Once the demonstration data is prepared by one of these three methods, instruction fine-tuning typically follows a similar training process as pre-training. The objective is to optimize the model to correctly predict each sequential token of the response. Depending on the situation, the model may have access to both the prompt and response or only the response during fine-tuning.

Instruction fine-tuning provides clear performance improvements. It also generally requires a much smaller dataset hence leaving a much smaller carbon footprint. For instance, Alpaca (Taori et al., 2023), fine-tuned using 52k self-generated instructions on LLaMA (Touvro et al., 2023), shows many behaviours similar to GPT-3 despite being surprisingly small (7B vs 176B) and cheap to reproduce (600$). Similarly, Flan-PaLM (Chung et al., 2022), fine-tuned using 1.8K tasks, outperforms PaLM by a large margin (+9.4% on average) while only requiring 0.2% of the pre-training compute.

Reinforcement learning from human feedback (RLHF)

Many studies (e.g., Kenton et al., 2021) have raised concerns that LLMs' behaviours are not well *aligned* with human goals. For example, they sometimes make up facts or generate biased and toxic text. Askell et al. (2021) proposed a framework defining aligned models as *helpful* (they should help the user solve their task), *honest* (they should not fabricate information or mislead the user), and *harmless* (they should not cause physical, psychological, or social harm to people or the environment).

One of the main sources of misalignment is the training objective itself. Both pre-training and instruction fine-tuning focus only on maximizing the likelihood of the next words instead of the quality of the entire response. To bridge this alignment gap, Ouyang et al. (2022) applied Reinforcement *Learning from Human Feedback (RLHF)* as an additional fine-tuning stage. It is considered by many one of the critical factors in the success of models like ChatGPT and GPT-4. RLHF consists of two key steps: *reward modeling (RM)* and *fine-tuning with reinforcement learning*.

Reward modeling

Reward modeling aims to assign a score to the genera
should reflect how well the response aligns with hun
human annotation work to gather a sufficiently large
model, and it can be difficult for subjective human annotators to agree on absolute scores.

responses. A human annotator is then asked to rank these responses from best to worst. Random pairs of responses are sampled to form the tuple (prompt, higher-rank response, lower-ranked response). This dataset is used to optimize the reward model by assigning scores that maximize the relative superiority of the higher-ranked response over the lower-ranked response. In this way, we can generate maximum $\binom{K}{2}$ tuples for each prompt with K responses (Figure 10a).

Fine-tuning with reinforcement learning

We have already seen how LLMs generate responses one token at a time. Each step depends on the previous output token, which was chosen using (controlled) random sampling. This poses a problem as gradient descent via backpropagation doesn't work here; a small change to the model parameters does not map to a small change in the output because of the intervening discrete sampling steps. This was not a problem for the next-word-prediction task, as sampling was not required. Hence, we need a different way to optimize the model.

The process of generating a response is similar to how a chess game is played. Both involve a series of discrete steps (generating tokens vs. making chess moves) and a quality assessment at the end (good or bad response of the reward model vs. win/lose/draw the game). Problems of this type have been well-explored in the field of *reinforcement learning,* where an AI agent optimized on the final reward signal (winning or losing the game) can perform far better than the best human player. To gain a detailed understanding of how reinforcement learning works, see Andrej Karpathy's blog.

It follows that reinforcement learning algorithms can naturally be adapted to optimize the quality of an LLM's responses. We start with a sampled prompt, which we use to get a corresponding response from the LLM. The response is assigned a score by the reward model built in the previous step. Finally, this score is used by a reinforcement learning algorithm like Proximal Policy Optimization (PPO) (John et al., 2017) to estimate the gradient update needed for optimizing the model (Figure 10b). The entire process is repeated to get the final fine-tuned LLM.

In Long et al.(2022), it is reported that human users significantly prefer responses generated from LLMs after RLHF (85 ± 3% of the time). However, while alignment fine-tuning helps guide the model toward generating human-like content, it can also be seen as adding additional constraints, which is called an "alignment tax" Thus, a careful balance needs to be struck between the model's capacity and alignment level.

Notable LLMs

The landscape of LLMs has rapidly evolved since 2020. These models range from web interfaces and APIs to completely accessible datasets, codebases, and model checkpoints. It is important to note that the terms for their commercial usage also differ. In this section, we highlight notable LLM models in chronological order, showcasing their unique features and contributions.

GPT-3 [API] was released by OpenAI in June 2020. Th
and is considered one of the most important LLM mil
demonstrate strong *few-shot learning* capabilities.

performance. ChatGPT was specifically designed for conversational use, offering a more natural and engaging chatbot experience.

GPT-4 [API] was released in March 2023. In addition to handling text, it can also take images as input. GPT-4 outperformed ChatGPT on many tasks, including passing the Bar exam. It also demonstrated better performance at safety tests (OpenAI, 2023).

LLaMA [checkpoint] was released by Meta in Feb 2023. It is a group of models with different numbers of parameters (7B, 13B, 33B, and 65B). Unlike GPT-X models that are accessible only through APIs, LLaMA provides implementation details of the model architecture and checkpoints. It has become a fundamental resource for subsequent open-source research endeavours.

Alpaca [checkpoint] was released by Stanford in March 2023. Alpaca was fine-tuned on top of LLaMA-7B using data generated by GPT-3. Remarkably, Alpaca demonstrated comparable performance to much larger models like GPT-3 (175B) while being smaller and more cost-effective to reproduce (less than $600).

Claude[API] was released by Alphabet-backed company Anthropic in March 2023. By adopting the Constitutional AI method proposed by Yuntao et al. (2022), Claude is reported to generate less toxic, biased, and hallucinatory responses. In May 2023, Anthropic expanded Claude's context window from 9k to 100K. This is equivalent to around 75,000 words and allows the model to parse all the information from an entire book at once.

Falcon[checkpoint] was released by TII in May 2023. It is an open-source LLM with 40 billion parameters under the Apache 2.0 license. Falcon stands out due to the high quality of its training data, which includes 1,000 billion tokens from the RefinedWeb enhanced with curated corpora. At the time of writing, Falcon holds the top position on the Huggingface Open LLM Leaderboard.

Evaluating LLM performance

Evaluating LLM performance is challenging as it encompasses various factors such as language coherence, task-specific performance, reasoning ability, and toxicity levels. Each factor needs to be further broken down to become measurable. Many evaluation benchmarks have been introduced (see Chang et al., 2023):

initiatives on building a unified evaluation framework that encompasses different datasets and tasks (e.g., Huggingface open LLM leaderboard, Stanford HELM), allow users to gain a holistic understanding of model performance in a more accessible manner.

Performance comparison

The MMLU leaderboard compiled the average accuracy of LLMs on MMLU tasks from technical reports and research papers. At the time of writing, GPT-4 holds the top position on the leaderboard with an overall accuracy of 86.4% (Figure: 1). According to the Huggingface open LLM leaderboard, Falcon-40B-Instruct achieves the highest average score among open-source LLMs at the time of writing.

It is worth noting that LLM performance can vary significantly depending on factors such as the dataset, training process, and target scenario. LLMs that excel in one specific area commonly underperform in others. A prime example is 30B-Lazarus. This model performs exceptionally well on the TruthfulQA task, which evaluates whether model responses are factually correct (Figure 11a). However, it demonstrates relatively poor performance in acquiring cross-domain knowledge, as reflected in the MMLU benchmark (Figure 11b). Therefore, when selecting an LLM for downstream tasks, it is crucial to consider the granular performance of the models and their suitability for specific requirements.

Limitations of LLMs

OpenAI(2023) shared a sample list of twelve observed safety challenges, ranging from harmful content and cybersecurity to economic impact. Understanding these and other limitations is crucial before utilizing LLMs. By doing so, policymakers, industry leaders, researchers, and the public can collaborate effectively to mitigate the associated risks. This section explores some of the most important concerns, their implications, and the

Cost

coupled with the specialized talent required, make LLM technology inaccessible to most organizations. This poses the risk of monopolies and over-reliance on a few companies that may not fully disclose their training data, methods, or potential biases of their models.

Fortunately, many open-source models (e.g., RedPajama) have been released to compete with closed-source LLMs like ChatGPT. While pre-training open-source LLMs remains costly, they offer transparency and enable collaborative efforts to further improve techniques for building LLMs. Users also have the option to further tailor the model by fine-tuning with domain-specific data.

Privacy and security

Concerns have been raised regarding LLMs regurgitating information they were trained on (Carlini et al., 2021). This poses a significant risk as private information used for training LLMs can potentially be leaked to the public. Using or developing LLMs necessitates an understanding of this risk and the implementation of appropriate measures to protect sensitive data.

One approach to mitigating privacy breaches is the removal of personally identifiable information (PII). Certain rule-based methods can be employed to detect and eliminate PII, including names, addresses, and phone numbers (Laurençon et al., 2023). For enterprise use cases, companies should consider restricting model access to employees with proper clearance to safeguard sensitive business information.

Trustworthiness

There is no guarantee that the output of an LLM will always be factually correct. In fact, it is not uncommon for LLMs to generate nonsensical or untruthful content. Such outputs are referred to as *hallucinations*. As LLMs become increasingly convincing, users may trust them to provide accurate information.

The adoption of reinforcement learning from human feedback (RLHF) for training models such as GPT-4 has reduced the frequency of hallucinations. OpenAI (2023) reports that GPT-4 scores 19 percentage points higher than the improved version of ChatGPT in evaluations of truthfulness. Other methods are also being explored to further mitigate this risk, such as providing links to external sources for cross-checking information or leveraging knowledge graphs to encourage LLMs to generate responses that are factually correct (Pan et al., 2023).

Copyright

Copyright has become a contentious point between [illegible] applies not only to LLMs but also to models generating

Copyright issues need to be considered from two perspectives: inputs and outputs. In terms of inputs, the question is whether or not mining and analyzing data for the purpose of training AI models infringes copyright. Quintais (2023) pointed out that such training data might fall under existing text and data mining exemptions. In terms of outputs, a number of copyright questions need to be answered. For example, is an output from an LLM protected by copyright? Does such an output infringe on a copyrighted work from a third party? These questions are currently being litigated in the US and the UK (e.g., the lawsuits brought by Getty Images). The outcomes of these cases, along with the European Union's requirements, will shape the legislation surrounding the output of LLMs.

How to apply LLMs without retraining from scratch

Harnessing the power of large language models doesn't necessarily require training them from scratch, which can be technically complex and computationally demanding. There are alternative approaches that allow users to leverage these models with fewer resources. In the following sections, we describe four options: *prompt engineering*, *prompt tuning*, using an LLM embedding as input, and model fine-tuning (Figure 12).

Prompt engineering

By providing appropriate prompts, users can utilize LLMs for tasks such as question answering (Q&A), language translation, or generating news articles (e.g., Awesome ChatGPT Prompts). The natural language interface makes prompting a versatile technique accessible to users without a machine learning background. Beyond that, since it is easy to access a variety of LLMs through tools such as LangChain and Hugging Face, prompting becomes the first stop of many users who want to leverage LLMs for specific use cases.

The effectiveness of LLMs is highly dependent on the quality of the prompts. For example, Brown et al. (2020) reported that giving a few examples as part of the prompt can provide a clear performance boost across 42 benchmarks. The term *prompt engineering* refers to optimizing the words used in prompts (*hard prompts*) to get the desired responses from LLMs (Figure 12a).

While there is no single template that works for all tasks, there are some basic components a prompt usually contains. Figure 13 shows one example following the 5-component template suggested by Microsoft's Azure OpenAI Service. This template serves as a good starting point for creating prompts.

After conducting extensive experiments, researchers have shared best practices for getting desired responses from LLMs (e.g., Open AI, Prompt Engineering Guide). For example, a prompt should be as specific as possible and show the output format requirements explicitly. These can be used for further polishing the prompt one is building.

As pointed out by Weng (2023), while there are many engineering, most of them are tricks that can be sumr

To help understand these ideas more intuitively, we have illustrated the difference among the ideas above (Figure 14). Since some of the techniques are orthogonal to each other (e.g., Self-consistency and CoT), they can be combined for better performance.

Prompt tuning

gradients and pass back the error signal to the prompt level. Then it updates the prompt accordingly (Figure 12b).

As long as model responses are desirable, there is no absolute need for prompts to be interpretable. Hence, instead of optimizing words used in a prompt, we can also optimize the token embeddings that are fed to the model. These token embeddings are referred to as *soft prompts*.

Lester et al. (2021) proposed a method to create soft prompts by adding token embeddings as the prefix to the original prompt's embedding. The combined sequence is fed into the model to get the final response. Prompt tuning only requires updating the additional soft prompt token embeddings, which is efficient relative to full fine-tuning but achieves the same level of performance on language understanding tasks when the LLM has enough parameters.

LLM embeddings as input

In addition to providing APIs for generating text-based responses from LLMs, companies like OpenAI also offer APIs to retrieve output embeddings from LLMs that have been fine-tuned for representation (see Neelakantan et al., 2022). This permits a new way of utilizing LLMs by extracting these embeddings and using them as inputs for downstream models. This approach has shown promising results in tasks such as text classification and semantic search.

Model fine-tuning

Prior research has highlighted the potential for further improvement when utilizing pretrained LLMs in specific domains. In the medical field, fine-tuning using domain-specific datasets such as Med-PaLM 2 exceeded the performance of GPT-4 on most medical Q\&A benchmarks (Singhal et al., 2023). Furthermore, fine-tuning an in-house LLM allows users to have better control over the model and avoid sharing sensitive business data with third parties.

However, traditional full fine-tuning approaches involve updating billions of parameters in the LLM, which can be memory-intensive and environmentally costly. Hence, there is growing interest in more parameter-efficient fine-tuning (PEFT) methods that can achieve comparable results.

One approach to improve efficiency is by incorporating adapters (e.g., Houlsby et al., 2019). While there are many variants of adapters, the typically involve adding additional trainable layers within a Transformer block (Figure: 15a). Another method is to append a sequence of trainable continuous vectors, also called as prefixes, to each Transformer layer(e.g., Liu et al., 2022) (Figure: 15b).

At the time of writing, one of the most widely adopted ap
(LoRA) (Hu et al., 2021). LoRA tackles the efficiency c
decomposition to represent the incremental updates
allows LoRA to focus exclusively on optimizing the inc

without introducing latency during inference. Unlike the appending prefixes approach, LoRA also does not require shrinking the available context window length. Furthermore, it can support multiple downstream tasks by only storing the incremental updates for each task. These characteristics make LoRA a promising solution for achieving improved efficiency in fine-tuning LLMs.

Discussion

In this blog, we have provided a high-level overview of large language models (LLMs). We discussed the model architecture (decoder-only Transformer), the three-step training process (pretraining, instruction fine-tuning, reinforcement learning from human feedback), representative LLMs (e.g., ChatGPT, LLaMA, Falcon), performance evaluation, limitations of LLMs (e.g., cost, hallucination), and different ways of exploiting LLMs (prompt engineering, prompt tuning, using LLM embeddings as input, and model fine-tuning). We intentionally provided a high-level perspective and omitted historical information, mathematical formulas, and less significant technical details.

It is important to note that the community is still in the early stages of adopting LLMs for broader use, and the technologies surrounding them are rapidly evolving. Hence, we conclude by highlighting some recent research directions that have caught our attention.

High-quality data and smaller models In June 2023, Microsoft released phi-1 (Gunasekar et al., 2023), a new large language model for code with only 1.3B parameters trained with 7B tokens. Despite the small number of parameters and small training dataset, it delivers surprisingly good performance across a list of benchmarks. This suggests that high-quality data might help break the existing scaling laws.

More accessible and efficient fine-tuning methods Researchers have investigated extensions and variations of LoRA. For example, QLoRA (Dettmers et al., 2023) applies quantization tricks to further reduce the memory requirements and computational cost. AdaLoRA (Zhang et al., 2023) adaptively allocates the parameter budget by using singular value decomposition. These methods further improve the efficiency of fine-tuning LLMs and have made fine-tuning more accessible to users with limited computational resources.

Extremely long context window length Until early 2023, the typical context window length of LLMs ranged from a few thousand to a few hundred thousand. However, in July 2023, Microsoft's LongNet Ding et al. (2023) pushed the context window length to an astonishing one billion tokens. This dramatic improvement opens new possibilities of using LLMs for use cases that require understanding extremely long sequences, or even the entire corpus, all at once.

To conclude, the field of LLMs is rapidly evolving, and c
various areas. It's essential to stay up-to-date with th
understand and leverage the full potential of LLMs. Fu
below.

BOREALIS AI

Transformer architecture

LLM survey

LLM evaluation and leaderboard

Tracking newly released LLMs

Learn prompt engineering

Instruction fine-tuning

Reinforcement learning and RLHF

Productionalization of LLMs

Acknowledgements

We would like to extend gratitude and appreciation towards Melissa Stabner, who played an essential role in assisting us by overseeing schedules, handling communication, offering valuable insights and implementing them effectively for marketing our work. We also want to thank April Cooper and Katrina Lowther, who went through many rounds of adjustment to create and improve all the visually appealing graphics used in this blog post.