

Chapter I: Introduction to LLMs

What are Large Language Models

By now, you might have heard of them. Large Language Models, commonly known as LLMs, are a **sophisticated type of neural network**. These models ignited many innovations in the field of natural language processing (NLP) and are **characterized by their large number of parameters, often in billions**, that make them proficient at processing and generating text. They are **trained on extensive textual data**, enabling them to grasp various language patterns and structures. The primary goal of LLMs is to **interpret and create human-like text that captures the nuances of natural language, including syntax (the arrangement of words) and semantics (the meaning of words)**.

The **core training objective** of LLMs focuses on **predicting the next word** in a sentence. This **straightforward objective** leads to the development of **emergent abilities**. For example, they **can conduct arithmetic calculations, unscramble words**, and have even demonstrated proficiency in professional exams, such as passing **the US Medical Licensing Exam**. Additionally, these models have significantly contributed to various NLP tasks, including machine translation, natural language generation, part-of-speech tagging, parsing, information retrieval, and others, even without direct training or fine-tuning in these specific areas.

The **text generation process** in Large Language Models is **autoregressive**, meaning **they generate the next tokens based on the sequence of tokens already generated**. The **attention mechanism** is a vital component in this process; it **establishes word connections** and **ensures the text is coherent and contextually appropriate**. It is essential to establish the fundamental terminology and concepts associated with Large Language Models before exploring the architecture and its building blocks (like attention mechanisms) in greater depth. Let's start with an overview of the architecture that powers these models, followed by defining a few terms, such as language modeling and tokenization.

Key LLM Terminologies

The Transformer

The foundation of a language model that makes it powerful lies in its architecture. Recurrent Neural Networks (RNNs) were traditionally used for text processing due to their ability to process sequential data. They maintain an internal state that retains information from previous words, facilitating sequential understanding. However, RNNs encounter challenges with long sequences where they forget older information in favor of recently processed input. This is primarily caused by the vanishing gradient problem, a phenomenon where the gradients, which are used to update the network's weights during training, become increasingly smaller as they are propagated back through each timestep of the sequence. As a result, the weights associated with early inputs change very little, hindering the network's ability to learn from and remember long-term dependencies within the data.

Transformer-based models addressed these challenges and emerged as the preferred architecture for natural language processing tasks. This architecture introduced in the influential paper “Attention Is All You Need” is a pivotal innovation in natural language processing. It forms the foundation for cutting-edge models like GPT-4, Claude, and LLaMA. The architecture was originally designed as an encoder-decoder framework. This setting uses an encoder to process input text, identifying important parts and creating a representation of the input. Meanwhile, the decoder is capable of transforming the encoder's output, a vector of high dimensionality, back into readable text for humans. These networks can be useful in tasks such as summarization, where the decoder generates summaries conditioned based on the articles passed to the encoder. It offers additional flexibility across a wide range of tasks since the components of this architecture, the encoder, and decoder, can be used jointly or independently. Some models use the encoder part of the network to transform the text into a vector representation or use only the decoder block, which is the backbone of the Large Language Models. The next chapter will cover each of these components.

Language Modeling

With the rise of LLMs, language modeling has become an essential part of natural language processing. It means learning the probability distribution of words within a language based on a large corpus. This learning process typically involves predicting the next token in a sequence using either classical statistical methods or novel deep learning techniques.

Large language models are trained based on the same objective to predict the next **word, punctuation mark, or other elements** based on the seen tokens in a text. These models become proficient by **understanding the distribution of words** within their training data by **guessing the probability of the next word based on the context**. For example, the model can complete a sentence beginning with “I live in New” with a word like “York” rather than an unrelated word such as “shoe”.

In practice, the **models work with tokens, not complete words**. This approach allows for more accurate predictions and text generation by more effectively capturing the complexity of human language.

Tokenization

Tokenization is the initial phase of interacting with LLMs. It involves **breaking down the input text into smaller pieces known as tokens**. Tokens can range from **single characters to entire words**, and the **size of these tokens can greatly influence the model’s performance**. Some models adopt **subword tokenization**, breaking words into smaller segments that retain meaningful linguistic elements.

Consider the following sentence, “The child’s coloring book.”

If tokenization splits the text after **every white space** character. The result will be:

```
["The", "child's", "coloring", "book."]
```

In this approach, you'll notice that the punctuation remains attached to the words like "child's" and "book."

Alternatively, tokenization can be done by separating text based on both white spaces and punctuation; the output would be:

```
["The", "child", "'", "s", "coloring", "book", "."]
```

The tokenization process is model-dependent. It's important to remember that the models are released as a pair of pre-trained tokenizers and associated model weights. There are more advanced techniques, like the Byte-Pair encoding, which is used by most of the recently released models. As demonstrated in the example below, this method also divides a word such as "coloring" into two parts.

```
["The", "child", "'", "s", "color", "ing", "book", "."]
```

Subword tokenization further enhances the model's language understanding by splitting words into meaningful segments, like breaking "coloring" into "color" and "ing." This expands the model's vocabulary and improves its ability to grasp the nuances of language structure and morphology. Understanding that the "ing" part of a word indicates the present tense allows us to simplify how we represent words in different tenses. We no longer need to keep separate entries for the base form of a word, like "play," and its present tense form, "playing." By combining "play" with "ing," we can express "playing" without needing two separate entries. This method increases the number of tokens to represent a piece of text but dramatically reduces the number of tokens we need to have in the dictionary.

The tokenization process involves scanning the entire text to identify unique tokens, which are then indexed to create a dictionary. This dictionary assigns a unique token ID to each token, enabling a standardized numerical representation of the text. When interacting with the models, this conversion of text into token IDs allows the model to efficiently process and understand the input, as it can quickly reference the dictionary to decode the meaning of each token. We will see an example of this process later in the book.

Once we have our tokens, we can process the inner workings of transformers: embeddings.

Embeddings

The next step after tokenization is to turn these tokens into something the computer can understand and work with—this is where embeddings come into play. Embeddings are a way to translate the tokens, which are words or pieces of words, into a language of numbers that the computer can grasp. They help the model understand relationships and context. They allow the model to see connections between words and use these connections to understand text better, mainly through the attention process, as we will see.

An embedding gives each token a unique numerical ID that captures its meaning. This numerical form helps the computer see how similar two tokens are, like knowing that “happy” and “joyful” are close in meaning, even though they are different words.

This step is essential because it helps the model make sense of language in a numerical way, bridging the gap between human language and machine processing.

Initially, every token is assigned a random set of numbers as its embedding. As the model is trained—meaning as it reads and learns from lots of text—it adjusts these numbers. The goal is to tweak them so that tokens with similar meanings end up with similar sets of numbers. This adjustment is done automatically by the model as it learns from different contexts in which the tokens appear.

While the concept of numerical sets, or vectors, might sound complex, they are just a way for the model to store and process information about tokens efficiently. We use vectors because they are a straightforward method for the model to keep track of how tokens are related to each other. They are basically just large lists of numbers.

In Chapter 2, we’ll explore more about how these embeddings are created and used in the transformer architecture.

Training/Fine-Tuning

LLMs are trained on a large corpus of text with the objective of correctly predicting the next token of a sequence. As we learned in the previous language modeling subsection, the goal is to adjust the model's parameters to maximize the probability of a correct prediction based on the observed data. Typically, a model is trained on a huge general-purpose dataset of texts from the Internet, such as The Pile or CommonCrawl. Sometimes, more specific datasets, such as the StackOverflow Posts dataset, are also an example of acquiring domain-specific knowledge. This phase is also known as the pre-training stage, indicating that the model is trained to learn language comprehension and is prepared for further tuning.

The training process adjusts the model's weights to increase the likelihood of predicting the next token in a sequence. This adjustment is based on the training data, guiding the model towards accurate token predictions.

After pre-training, the model typically undergoes fine-tuning for a specific task. This stage requires further training on a smaller dataset for a task (e.g., text translation) or a specialized domain (e.g., biomedical, finance, etc.). Fine-tuning allows the model to adjust its previous knowledge of the specific task or domain, enhancing its performance.

The fine-tuning process can be intricate, particularly for advanced models such as GPT-4. These models employ advanced techniques and leverage large volumes of data to achieve their performance levels.

Prediction

The model can generate text after the training or fine-tuning phase by predicting subsequent tokens in a sequence. This is achieved by inputting the sequence into the model, producing a probability distribution over the potential next tokens, essentially assigning a score to every word in the vocabulary. The next token is selected according to its score. The generation process will be repeated in a loop to predict one word at a time, so

generating sequences of any length is possible. However, keeping the model's effective context size in mind is essential.

Context Size

The context size, or context window, is a crucial aspect of LLMs. It refers to the maximum number of tokens the model can process in a single request. Context size influences the length of text the model can handle at any one time, directly affecting the model's performance and the outcomes it produces.

Different LLMs are designed with varying context sizes. For example, OpenAI's "gpt-3.5-turbo-16k" model has a context window capable of handling 16,000 tokens. There is an inherent limit to the number of tokens a model can generate. Smaller models may have a capacity of up to 1,000 tokens, while larger ones like GPT-4 can manage up to 32,000 tokens as of the time we wrote this book.

Scaling Laws

Scaling laws describe the relationship between a language model's performance and various factors, including the number of parameters, the training dataset size, the compute budget, and the network architecture. These laws, elaborated in the [Chinchilla paper](#), provide useful insights on resource allocation for successful model training. They are also a source of many memes from the "scaling is all you need" side of the community in AI.

The following elements determine a language model's performance:

1. The number of parameters (N) denotes the model's ability to learn from data. A greater number of parameters enables the detection of more complicated patterns in data.
2. The size of the Training Dataset (D) and the number of tokens, ranging from small text chunks to single characters, are counted.

3. FLOPs (Floating Point Operations Per Second) estimate the computational resources used during training.

In their research, the authors trained the Chinchilla model, which comprises 70 billion parameters, on a dataset of 1.4 trillion tokens. This approach aligns with the scaling law proposed in the paper: **for a model with X parameters, the optimal training involves approximately $X * 20$ tokens**. For example, a model with 100 billion parameters would ideally be trained on about 2 trillion tokens.

With this approach, despite its smaller size compared to other LLMs, the Chinchilla model outperformed them all. It improved language modeling and task-specific performance using less memory and computational power. Find the paper “Training Compute-Optimal Large Language Models.” at towardsai.net/book.

Emergent Abilities in LLMs

Emergent abilities in LLMs describe the phenomena in which new skills emerge unexpectedly as model size grows. These abilities, including arithmetic, answering questions, summarizing material, and others, are not explicitly taught to the model throughout its training. Instead, they emerge spontaneously when the model’s scaling increases, hence the word “emergent.”

LLMs are probabilistic models that learn natural language patterns. When these models are ramped up, their pattern recognition capacity improves quantitatively while also changing qualitatively.

Traditionally, models required task-specific fine-tuning and architectural adjustments to execute specific tasks. However, scaled-up models can perform these jobs without architectural changes or specialized tuning. They accomplish this by interpreting tasks using natural language processing. LLMs’ ability to accomplish various functions without explicit fine-tuning is a significant milestone.

What's more remarkable is how these abilities show themselves. LLMs swiftly and unpredictably progress from near-zero to sometimes state-of-the-art performance as their size grows. This phenomenon indicates that these abilities arise from the model's scale rather than being clearly programmed into the model.

This growth in model size and the expansion of training datasets, accompanied by substantial increases in computational costs, paved the way for the emergence of today's Large Language Models. Examples of such models include Cohere Command, GPT-4, and LLaMA, each representing significant milestones in the evolution of language modeling.

Prompts

The text (or images, numbers, tables...) we provide to LLMs as instructions is commonly called prompts. Prompts are instructions given to AI systems like OpenAI's GPT-3 and GPT-4, providing context to generate human-like text—the more detailed the prompt, the better the model's output.

Concise, descriptive, and short (depending on the task) prompts generally lead to more effective results, allowing for the LLM's creativity while guiding it toward the desired output. Using specific words or phrases can help focus the model on generating relevant content. Creating effective prompts requires a clear purpose, keeping things simple, strategically using keywords, and assuring actionability. Testing prompts before final use is critical to ensure the output is relevant and error-free. Here are some prompting tips:

1. **Use Precise Language:** Precision in your prompt can significantly improve the accuracy of the output.
 - Less Precise: "Write about dog food."
 - More Precise: "Write a 500-word informative article about the dietary needs of adult Golden Retrievers."
1. **Provide Sufficient Context:** Context helps the model understand the expected output:
 - Less Contextual: "Write a story."

- **More Contextual:** “Write a short story set in Victorian England featuring a young detective solving his first major case.”
1. **Test Variations:** Experiment with different prompt styles to find the most effective approach:
 - **Initial:** “Write a blog post about the benefits of yoga.”
 - **Variation 1:** “Compose a 1000-word blog post detailing the physical and mental benefits of regular yoga practice.”
 - **Variation 2:** “Create an engaging blog post that highlights the top 10 benefits of incorporating yoga into a daily routine.”
 1. **Review Outputs:** Always double-check automated outputs for accuracy and relevance before publishing.
 - Before Review: “Yoga is a great way to improve your flexibility and strength. It can also help reduce stress and improve mental clarity. However, it’s important to remember that all yoga poses are suitable for everyone.”
 - After Review (corrected): “Yoga is a great way to improve your flexibility and strength. It can also help reduce stress and improve mental clarity. However, it’s important to remember that not all yoga poses are suitable for everyone. Always consult with a healthcare professional before starting any new exercise regimen.”

Hallucinations and Biases in LLMs

Hallucinations in AI systems refer to instances where these systems produce outputs, such as text or visuals, inconsistent with facts or the available inputs. One example would be if ChatGPT provides a compelling but factually wrong response to a question. These hallucinations show a mismatch between the AI’s output and real-world knowledge or context.

In LLMs, hallucinations occur when the model creates outputs that do not correspond to real-world facts or context. This can lead to the spread of disinformation, especially in crucial industries like

healthcare and education, where information accuracy is critical. Bias in LLMs can also result in outcomes that favor particular perspectives over others, possibly reinforcing harmful stereotypes and discrimination.

An example of a hallucination could be if a user asks, “Who won the World Series in 2025?” and the LLM responds with a specific winner. As of the current date (Jan 2024), the event has yet to occur, making any response speculative and incorrect.

Additionally, **Bias** in AI and LLMs is another critical issue. It refers to these models’ inclination to favor specific outputs or decisions based on their training data. If the training data primarily originates from a particular region, the model may be biased toward that region’s language, culture, or viewpoints. In cases where the training data encompasses biases, like gender or race, the resulting outputs from the AI system could be biased or discriminatory.

For example, if a user asks an LLM, “Who is a nurse?” and it responds, “She is a healthcare professional who cares for patients in a hospital,” this demonstrates a gender bias. The paradigm inherently associates nursing with women, which needs to adequately reflect the reality that both men and women can be nurses.

Mitigating hallucinations and bias in AI systems involves refining model training, using verification techniques, and ensuring the training data is diverse and representative. Finding a balance between maximizing the model’s potential and avoiding these issues remains challenging.

Amazingly, these “hallucinations” might be advantageous in creative fields such as fiction writing, allowing for the creation of new and novel content. The ultimate goal is to create powerful, efficient but also trustworthy, fair, and reliable LLMs. We can maximize the promise of LLMs while minimizing their hazards, ensuring that the advantages of this technology are available to all.

Translation with LLMs (GPT-3.5 API)

Now, we can combine all we have learned to demonstrate how to interact with OpenAI's proprietary LLM through their API, instructing the model to perform translation. To generate text using LLMs like those provided by OpenAI, you first need an API key for your Python environment. Here's a step-by-step guide to generating this key:

1. Create and log into your OpenAI account.
2. After logging in, select 'Personal' from the top-right menu and click "View API keys."
3. You'll find the "Create new secret key" button on the API keys page. Click on it to generate a new secret key. Remember to save this key securely, as it will be used later.

After generating your API key, you can securely store it in a `.env` file using the following format:

```
OPENAI_API_KEY="<YOUR-OPENAI-API-KEY>"
```

Every time you initiate a Python script including the following lines, your API key will be automatically loaded into an environment variable named `OPENAI_API_KEY`. The `openai` library subsequently uses this variable for text generation tasks. The `.env` file must be in the same directory as the Python script.

```
from dotenv import load_dotenv  
  
load_dotenv()
```

Now, the model is ready for interaction! Here's an example of using the model for a language translation from English to French. The code below sends the prompt as a message with a user role, using the OpenAI Python package to send and retrieve requests from the API. There is no need for concern if you do not understand all the details, as we will use the OpenAI API more thoroughly in Chapter 5. It would be best if you focused on the

messages argument for now, which receives the prompt that directs the model to execute the translation task.

```
from dotenv import load_dotenv
load_dotenv()
import os
import openai

# English text to translate
english_text = "Hello, how are you?"

response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": f'''Translate the following English text
to French: "{english_text}'''
    ]
)

print(response['choices'][0]['message']['content'])
```

```
Bonjour, comment ça va?
```

💡 You can safely store sensitive information, such as API keys, in a separate file with `dotenv` and avoid accidentally exposing it in your code. This is especially important when working with open-source projects or sharing your code with others, as it ensures the security of sensitive information.

Control LLMs Output by Providing Examples

Few-shot learning, which is one of the emergent abilities of LLMs, means providing the model with a small number of examples before making predictions. These examples serve a dual purpose: they “teach” the model in its reasoning process and act as “filters,” aiding the model in identifying relevant patterns within its dataset. Few-shot learning allows for the adaptation of the model to new tasks. While LLMs like GPT-3 show proficiency in language modeling tasks such as machine translation, their performance can vary on tasks that require more complex reasoning.

In few-shot learning, the examples presented to the model help discover relevant patterns in the dataset. The datasets are effectively encoded into the model's weights during the training, so the model looks for patterns that significantly connect with the provided samples and uses them to generate its output. As a result, the model's precision improves by adding more examples, allowing for a more targeted and relevant response.

Here is an example of few-shot learning, where we provide examples through different message types on how to describe movies with emojis to the model. (We will cover the different message types later in the book.) For instance, the movie "Titanic" might be presented using emojis for a cruise ship, waves, a heart, etc., or how to represent "The Matrix" movie. The model picks up on these patterns and manages to accurately describe the movie "Toy Story" using emojis of toys.

```
from dotenv import load_dotenv
load_dotenv()
import os
import openai

# Prompt for summarization
prompt = """
Describe the following movie using emojis.

{movie}: """

examples = [
    { "input": "Titanic", "output": "🚢🌊❤️💻🎵🔥🚢💔👫🎭" },
    { "input": "The Matrix", "output": "🕶️💊💥🤖💡🏙️👤🔄🔒💪" }
]

movie = "Toy Story"
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": prompt.format(movie=examples[0]
["input"])}},
        {"role": "assistant", "content": examples[0]["output"]},
        {"role": "user", "content": prompt.format(movie=examples[1]
["input"])}},
        {"role": "assistant", "content": examples[1]["output"]},
        {"role": "user", "content": prompt.format(movie=movie)},
```



```
]
)
print(response['choices'][0]['message']['content'])
```



It's fascinating how the model, with just two examples, can identify a complex pattern, such as associating a film title with a sequence of emojis. This ability is achievable only with a model that possesses an in-depth understanding of the film's story and the meaning of the emojis, allowing it to merge the two and respond to inquiries based on its own interpretation.

From Language Models to Large Language Models

The evolution of language models has seen a paradigm shift from pre-trained language models (LMs) to the creation of Large Language Models (LLMs). LMs, such as ELMo and BERT, first captured context-aware word representations through pre-training and fine-tuning for specific tasks. However, the introduction of LLMs, as demonstrated by GPT-3 and PaLM, proved that scaling model size and data can unlock emergent skills that outperform their smaller counterparts. Through in-context learning, these LLMs can handle more complex tasks.

Emergent Abilities in LLMs

As we discussed, an ability is considered emergent when larger models exhibit it, but it's absent in smaller models—a key factor contributing to the success of Large Language Models. **Emergent abilities** in Large Language Models (LLMs) are empirical phenomena that occur when the size of language models exceeds specific thresholds. As we increase the models' size, emergent abilities become more evident, influenced by aspects like the computational power used in training and the model's parameters.

What Are Emergent Abilities

This phenomenon indicates that the models are learning and generalizing beyond their pre-training in ways that were not explicitly programmed or anticipated. A distinct pattern emerges when these abilities are depicted on a scaling curve. Initially, the model's performance appears almost random, but it significantly improves once a certain scale threshold is reached. This phenomenon is known as a phase transition, representing a dramatic behavior change that would not have been apparent from examining smaller-scale systems.

Scaling language models have predominantly focused on increasing the amount of computation, expanding the model parameters, and enlarging the training dataset size. New abilities can sometimes emerge with reduced training computation or fewer model parameters, especially when models are trained on higher-quality data. Additionally, the appearance of emergent abilities is influenced by factors such as the volume and quality of the data and the quantity of the model's parameters. Emergent abilities in Large Language Models surface as the models are scaled up and are not predictable by merely extending the trends observed in smaller models.

Evaluation Benchmarks for Emergent Abilities

Several benchmarks are used to evaluate the emergent abilities of language models, such as BIG-Bench, TruthfulQA, the Massive Multi-task Language Understanding (MMLU) benchmark, and the Word in Context (WiC) benchmark. Key benchmarks include:

1. **BIG-Bench suite** comprises over 200 benchmarks testing a wide array of tasks, such as arithmetic operations (example: “Q: What is 132 plus 762? A: 894), transliteration from the International Phonetic Alphabet (IPA), and word unscrambling. These tasks assess a model's capacity to perform calculations, manipulate and use rare words, and work with alphabets. (example: “English: The 1931 Malay census was an alarm bell. IPA: ðə 1931 'meɪleɪ 'sɛnsəs wəz ən ə'lɑrm bɛl.”) The performance of models like GPT-3 and LaMDA on these tasks usually starts near

zero but shows a significant increase above random at a certain scale, indicative of emergent abilities. More details on these benchmarks can be found in the Github repository.

2. **TruthfulQA** benchmark evaluates a model's ability to provide truthful responses. It includes two tasks: generation, where the model answers a question in one or two sentences, and multiple-choice, where the model selects the correct answer from four options or True/False statements. As the Gopher model is scaled to its largest size, its performance improves significantly, exceeding random outcomes by over 20%, which signifies the emergence of this ability.
3. **Massive Multi-task Language Understanding (MMLU)** assesses a model's world knowledge and problem-solving skills across 57 diverse tasks, including elementary mathematics, US history, and computer science. While GPTs, Gopher, and Chinchilla models of a certain scale do not outperform random guessing on average across all topics, a larger size model shows improved performance, suggesting the emergence of this ability.
4. The **Word in Context (WiC)** benchmark focuses on semantic understanding and involves a binary classification task for context-sensitive word embeddings. It requires determining if target words (verbs or nouns) in two contexts share the same meaning. Models like Chinchilla initially fail to surpass random performance in one-shot tasks, even at large scales. However, when models like PaLM are scaled to a much larger size, above-random performance emerges, indicating the emergence of this ability at a larger scale.

Factors Leading To Emergent Abilities

- Multi-step reasoning involves instructing a model to perform a series of intermediate steps before providing the final result. This approach, known as **chain-of-thought prompting**, becomes more effective than standard prompting only when applied to sufficiently large models.

- Another strategy is **fine-tuning a model** on various tasks presented as **Instruction Following**. This method shows improved performance only with models of a certain size, underlining the significance of scale in achieving advanced capabilities.

Risks With Emergent Abilities

As language models are scaled up, **emergent risks** also become a concern. These include **societal challenges related to accuracy, bias, and toxicity**. **Adopting strategies that encourage models to be “helpful, harmless, and honest” can mitigate** these risks.

For instance, the **WinoGender benchmark**, which assesses gender bias in occupational contexts, has shown that while **scaling can enhance model performance**, it may **also amplify biases, especially in ambiguous situations**. Larger models tend to **memorize training data more**, but methods like **deduplication can reduce this risk**.

Other risks involve potential vulnerabilities or harmful content synthesis that might be more prevalent in future language models or remain uncharacterized in current models.

A Shift Towards General-Purpose Models

The emergence of new abilities has shifted the NLP community’s perspective and utilization of these models. While NLP traditionally focused on task-specific models, the scaling of models has spurred research on “general-purpose” models capable of handling a wide range of tasks not explicitly included in their training.

This shift is evident in instances where scaled, **few-shot prompted general-purpose models have outperformed task-specific models that were fine-tuned**. Examples include GPT-3 setting new benchmarks in **TriviaQA and PiQA**, PaLM excelling in **arithmetic reasoning**, and the multimodal **Flamingo model** achieving top performance in **visual question answering**. Furthermore, the ability of general-purpose models to execute tasks with minimal examples has expanded their applications beyond traditional NLP

research. These include translating natural language instructions for robotic execution, user interaction, and multi-modal reasoning.

Expanding the Context Window

The Importance of Context Length

Context window in language models represents the number of input tokens the model can process simultaneously. In models like GPT-4, it currently stands at approximately 32K or roughly 50 pages of text. However, recent advancements have extended this to an impressive 100K tokens or about 156 pages, as seen in Claude by Anthropic.

Context length primarily enables the model to process and comprehend larger datasets simultaneously, offering a deeper understanding of the context. This feature is particularly beneficial when inputting a substantial amount of specific data into a language model and posing questions related to this data. For example, when analyzing a lengthy document about a particular company or issue, a larger context window allows the language model to review and remember more of this unique information, resulting in more accurate and tailored responses.

Limitations of the Original Transformer Architecture

Despite its strengths, the original transformer architecture faces challenges in handling extensive context lengths. Specifically, the attention layer operations in the transformer have quadratic time and space complexity (represented with $O(n^2)$) in relation to the number of input tokens, n . As the context length expands, the computational resources required for training and inference increase substantially.

To better understand this, let's examine the computational complexity of the transformer architecture. The complexity of the attention layer in the transformer model is $O(n^2d)$, where n is the context length (number of input tokens) and d is the embedding size.

This complexity stems from two primary operations in the attention layer: linear projections to create Query, Key, and Value matrices (complexity \sim) and the multiplication of these matrices (complexity \sim). As the context length or embedding size increases, the computational complexity also grows quadratically, presenting a challenge for processing larger context lengths.

Optimization Techniques to Expand the Context Window

Despite the computational challenges associated with the original transformer architecture, researchers have developed a range of optimization techniques to enhance the transformer's efficiency and increase its context length capacity to 100K tokens:

1. **ALiBi Positional Encoding:** The original transformer used Positional Sinusoidal Encoding, which has trouble inferring larger context lengths. On the other hand, ALiBi (Attention with Linear Biases) is a more scalable solution. This positional encoding technique allows the model to be trained in smaller contexts and then fine-tuned in bigger contexts, making it more adaptive to different context sizes.
2. **Sparse Attention:** Sparse Attention addresses the computational challenge by focusing attention scores on a subset of tokens. This method significantly decreases the computing complexity to a linear scale with respect to the number of tokens n , resulting in a significant reduction in overall computational demand.
3. **FlashAttention:** FlashAttention restructures the attention layer calculation for GPU efficiency. It divides input matrices into blocks and then processes attention output with reference to these blocks, optimizing GPU memory utilization and increasing processing efficiency.
4. **Multi-Query Attention (MQA):** MQA reduces memory consumption in the key/value decoder cache by aggregating weights across all attention heads during linear projection of the

Key and Value matrices. This consolidation results in more effective memory utilization.

FlashAttention-2

FlashAttention-2 emerges as an advancement over the original FlashAttention, focusing on optimizing the speed and memory efficiency of the attention layer in transformer models. This upgraded version is redeveloped from the ground up utilizing Nvidia's new primitives. It performs approximately 2x faster than its predecessor, achieving up to 230 TFLOPs on A100 GPUs.

FlashAttention-2 improves on the original FlashAttention in various ways.

- Changing the algorithm to spend more time on matmul FLOPs minimizes the quantity of non-matmul FLOPs, which are 16x more expensive than matmul FLOPs.
- It optimizes parallelism across batch size, headcount, and sequence length dimensions, leading to significant acceleration, particularly for long sequences.
- It enhances task partitioning within each thread block to reduce synchronization and communication between warps, resulting in fewer shared memory reads/writes.
- It adds features such as support for attention head dimensions up to 256 and multi-query attention (MQA), further expanding the context window.

With these enhancements, FlashAttention-2 is a successful step toward context window expansion (while still retaining the underlying restrictions of the original transformer architecture).

LongNet: A Leap Towards Billion-Token Context Window

LongNet represents a transformative advancement in the field of transformer optimization, as detailed in the paper “[LONGNET: Scaling Transformers to 1,000,000,000 Tokens](#)”. This innovative approach is set to extend the context window of language models to an unprecedented 1 billion tokens, significantly enhancing their ability to process and analyze large volumes of data.

The primary advancement in LongNet is the implementation of “dilated attention.” This innovative attention mechanism allows for an exponential increase in the attention field as the gap between tokens widens, inversely reducing attention calculations as the distance between tokens increases. (since every token will attend to a smaller number of tokens). This design approach balances the limited attention resources and the need to access every token in the sequence.

LongNet’s dilated attention mechanism has a linear computational complexity, a major improvement over the normal transformer’s quadratic difficulty.

A Timeline of the Most Popular LLMs

Here’s the timeline of some of the most popular LLMs in the last five years.

- **[2018]** [GPT-1](#)

Introduced by OpenAI, GPT-1 laid the foundation for the GPT series with its generative, decoder-only transformer architecture. It pioneered the combination of unsupervised pretraining and supervised fine-tuning for natural language text prediction.

- **[2019]** [GPT-2](#)

Building on GPT-1’s architecture, GPT-2 expanded the model size to 1.5 billion parameters, demonstrating the model’s versatility across a range of tasks using a unified format for input, output, and task information.

- [2020]GPT-3

Released in 2020, GPT-3 marked a substantial leap with 175 billion parameters, introducing in-context learning (ICL). This model showcased exceptional performance in various NLP tasks, including reasoning and domain adaptation, highlighting the potential of scaling up model size.

- [2021]Codex

OpenAI introduced Codex in July 2021. It is a GPT-3 variant fine-tuned on a corpus of GitHub code and exhibited advanced programming and mathematical problem-solving capabilities, demonstrating the potential of specialized training.

- [2021]LaMDA

Researchers from DeepMind introduced LaMDA (Language Models for Dialog Applications). LaMDA focused on dialog applications, boasting 137 billion parameters. It aimed to enhance dialog generation and conversational AI.

- [2021]Gopher

In 2021, DeepMind's Gopher, with 280 billion parameters, approached human-level performance on the MMLU benchmark but faced challenges like biases and misinformation.

- [2022]InstructGPT

In 2022, InstructGPT, an enhancement to GPT-3, utilized reinforcement learning from human feedback to improve instruction-following and content safety, aligning better with human preferences

- [2022]Chinchilla

DeepMind's Chinchilla introduced in 2022, with 70 billion parameters, optimized compute resource usage based on scaling laws, achieving significant accuracy improvements on benchmarks.

- [2022]PaLM

Pathways Language Model (PaLM) was introduced by Google Research in 2022. Google's PaLM, with an astounding 540 billion parameters, demonstrated exceptional few-shot performance, benefiting from Google's Pathways system for distributed computation.

- [2022]ChatGPT

In November 2022, OpenAI's ChatGPT, based on GPT-3.5 and GPT-4, was tailored for conversational AI and showed proficiency in human-like communication and reasoning.

- [2023] [LLaMA](#)

Meta AI developed LLaMA (Large Language Model Meta AI) in February 2023. It introduced a family of massive language models with parameters ranging from 7 billion to 65 billion. The publication of LLaMA broke the tradition of limited access by making its model weights available to the scientific community under a noncommercial license. Subsequent innovations, such as [LLaMA 2](#) and other chat formats, stressed accessibility even further, this time with a commercial license.

- [2023] [GPT-4](#)

In March 2023, GPT-4 expanded its capabilities to multimodal inputs, outperforming its predecessors in various tasks and representing another significant step in LLM development.

- [2024] [Gemini 1.5](#)

Gemini 1.5 (from Google) features a significant upgrade compared to the previous iteration of the model with a new Mixture-of-Experts architecture and multimodal model capability. Gemini 1.5 Pro, which supports advanced long-context understanding and a context window of up to 1 million tokens. The context window size is larger than any other model available today. The model is accessible through Google's proprietary API.

- [2024] [Gemma](#)

Google has also released the Gemma model in two versions: 2 billion and 7 billion parameters. These models were developed during the training phase that produced the Gemini model and are now publicly accessible. Users can access these models in both pre-trained and instruction-tuned formats.

- [2024] [Claude 3 Opus](#)

The newest model from Anthropic, the Claude 3 Opus, is available via their proprietary API. It is one of the first models to achieve scores comparable to or surpassing GPT-4 across different benchmarks. With a context window of 200K tokens, it is advertised for its exceptional recall capabilities, regardless of the position of the information within the window.

- **[2024]**[Mistral](#)

Following their publication detailing the **Mixture of Experts** architecture, they have **now made the 8x22 billion base model** available to the public. This model is the **best open-source option** currently accessible for use. Despite this, it still does not outperform the performance of closed-source models like GPT-4 or Claude.

- **[2024]**[Infinite Attention](#)

Google's recent paper, speculated to be the base of the Gemini 1.5 Pro model, explores techniques that could indefinitely **expand the model's context window size**. Speculation surfaced because the paper released alongside the Gemini model mentioned that the model could perform exceptionally well **with up to 10 million tokens**. However, a model with these specifications has yet to be released. This approach is described as a plug-and-play solution that can significantly enhance any model's **few-shot learning performance without context size constraints**.

If you want to dive deeper into these models, we suggest reading the paper "[A Survey of Large Language Models](#)".

History of NLP/LLMs

This is a journey through the growth of language modeling models, from early statistical models to the birth of the first Large Language Models (LLMs). Rather than an in-depth technical study, this chapter presents a story-like exploration of model building. Don't worry if certain model specifics appear complicated.

The Evolution of Language Modeling

The evolution of natural language processing (NLP) models is a story of constant invention and improvement. The **Bag of Words model**, a simple approach for counting word occurrences in documents, began in **1954**. Then, in **1972**, **TF-IDF** appeared, improving on this strategy by altering word counts based on rarity or frequency. The introduction of **Word2Vec** in **2013** marked a significant breakthrough. This model used **word embeddings**

to capture subtle semantic links between words that previous models could not.

Following that, Recurrent Neural Networks (RNNs) were introduced. RNNs were adept at learning patterns in sequences, allowing them to handle documents of varied lengths effectively.

The launch of the transformer architecture in 2017 signified a paradigm change in the area. During output creation, the model's attention mechanism allowed it to focus on the most relevant elements of the input selectively. This breakthrough paved the way for BERT in 2018. BERT used a bidirectional transformer, significantly increasing performance in various traditional NLP workloads.

The years that followed saw a rise in model developments. Each new model, such as RoBERTa, XLM, ALBERT, and ELECTRA, introduced additional enhancements and optimizations, pushing the bounds of what was feasible in NLP.

Model's Timeline

- [1954]Bag of Words (BOW)

The Bag of Words model was a basic approach that tallied word occurrences in manuscripts. Despite its simplicity, it could not consider word order or context.

- [1972]TF-IDF

TF-IDF expanded on BOW by giving more weight to rare words and less to common terms, improving the model's ability to detect document relevancy. Nonetheless, it made no mention of word context.

- [2013]Word2Vec

Word embeddings are high-dimensional vectors encapsulating semantic associations, as described by Word2Vec. This was a substantial advancement in capturing textual semantics.

- [2014]RNNs in Encoder-Decoder architectures

RNNs were a significant advancement, capable of computing document embeddings and adding word context. They grew to include LSTM (1997) for long-term dependencies and Bidirectional RNN (1997) for context understanding. Encoder-Decoder RNNs (2014) improved on this method.

- [2017]Transformer

The transformer, with its attention mechanisms, greatly improved embedding computation and alignment between input and output, revolutionizing NLP tasks.

- [2018]BERT

BERT, a bidirectional transformer, achieved impressive NLP results using global attention and combined training objectives.

- [2018]GPT

The transformer architecture was used to create the first autoregressive model, GPT. It then evolved into GPT-2 [2019], a larger and more optimized version of GPT pre-trained on WebText, and GPT-3 [2020], a larger and more optimized version of GPT-2 pre-trained on Common Crawl.

- [2019]CTRL

CTRL, similar to GPT, introduced control codes enabling conditional text generation. This feature enhanced control over the content and style of the generated text.

- [2019]Transformer-XL

Transformer-XL innovated by reusing previously computed hidden states, allowing the model to maintain a longer contextual memory. This enhancement significantly improved the model's ability to handle extended text sequences.

- [2019]ALBERT

ALBERT offered a more efficient version of BERT by implementing Sentence Order Prediction instead of Next Sentence Prediction and employing parameter-reduction techniques. These changes resulted in lower memory usage and expedited training.

- [2019]RoBERTa

RoBERTa improved upon BERT by introducing dynamic Masked Language Modeling, omitting the Next Sentence Prediction, using the BPE tokenizer, and employing better hyperparameters for enhanced performance.

- [2019]XLM

XLM was a multilingual transformer, pre-trained using a variety of objectives, including Causal Language Modeling, Masked Language Modeling, and Translation Language Modeling, catering to multilingual NLP tasks.

- [2019]XLNet

XLNet combined the strengths of Transformer-XL with a generalized autoregressive pretraining approach, enabling the learning of bidirectional dependencies and offering improved performance over traditional unidirectional models.

- [2019]PEGASUS

PEGASUS featured a bidirectional encoder and a left-to-right decoder, pre-trained using objectives like Masked Language Modeling and Gap Sentence Generation, optimizing it for summarization tasks.

- [2019]DistilBERT

DistilBERT presented a smaller, faster version of BERT, retaining over 95% of its performance. This model was trained using distillation techniques to compress the pre-trained BERT model.

- [2019]XLM-RoBERTa

XLM-RoBERTa was a multilingual adaptation of RoBERTa, trained on a diverse multilanguage corpus, primarily using the Masked Language Modeling objective, enhancing its multilingual capabilities.

- [2019]BART

BART, with a bidirectional encoder and a left-to-right decoder, was trained by intentionally corrupting text and then learning to reconstruct the original, making it practical for a range of generation and comprehension tasks.

- [2019]ConvBERT

ConvBERT innovated by replacing traditional self-attention blocks with modules incorporating convolutions, allowing for more effective handling of global and local contexts within the text.

- [2020]Funnel Transformer

Funnel Transformer innovated by progressively compressing the sequence of hidden states into a shorter sequence, effectively reducing computational costs while maintaining performance.

- [2020]Reformer

Reformer offered a more efficient version of the transformer. It utilized locality-sensitive hashing for attention mechanisms and axial position encoding, among other optimizations, to enhance efficiency.

- [2020]T5

T5 approached NLP tasks as a text-to-text problem. It was trained using a mixture of unsupervised and supervised tasks, making it versatile for various applications.

- [2020]Longformer

Longformer adapted the transformer architecture for longer documents. It replaced traditional attention matrices with sparse versions, improving training efficiency and better handling of longer texts.

- [2020]ProphetNet

ProphetNet was trained using a Future N-gram Prediction objective, incorporating a unique self-attention mechanism. This model aimed to improve sequence-to-sequence tasks like summarization and question-answering.

- [2020]ELECTRA

ELECTRA presented a novel approach, trained with a Replaced Token Detection objective. It offered improvements over BERT in efficiency and performance across various NLP tasks.

- [2021]Switch Transformers

Switch Transformers introduced a sparsely-activated expert model, a new spin on the Mixture of Experts (MoE) approach. This design allowed the model to manage a broader array of tasks more efficiently, marking a significant step towards scaling up transformer models.

Recap

The advancements in natural language processing, beginning with the essential Bag of Words model, led us to the advanced and highly sophisticated transformer-based models we have today. Large language models (LLMs) are powerful architectures trained on massive amounts of