Large Language Models (LLMs): System Design and Applications

Alireza Dirafzoon, PhD (Ex-Meta, Ex-Samsung)

Preface

What is this book for?

This book aims to serve as a comprehensive guide to understanding and implementing Large Language Models (LLMs) in various fields. It is designed to bridge the gap between theoretical knowledge and practical applications, providing readers with a clear understanding of the design, functionality, and potential of LLMs. Through a mix of technical insights, case studies, and forward-looking discussions, this book seeks to empower professionals, researchers, and enthusiasts to effectively utilize LLMs in their respective domains.

Who should read this book?

The guide here is mostly focused on software engineers, data scientists, AI researchers, and technology enthusiasts who are keen to delve into the world of Large Language Models. It is also highly beneficial for educators and students in computer science and artificial intelligence, providing them with a detailed overview of the latest advancements and methodologies in LLMs. Business leaders and decision-makers looking to understand the impact of LLMs in their industries will also find this book valuable for strategic planning.

What does this book cover?

The following chapters of this book cover a wide range of topics related to Large Language Models. Starting with a historical overview of language models and their evolution, the book delves into the architecture, training processes, and technical nuances of current LLMs. We explore various applications and case studies, highlighting how LLMs are transforming industries from healthcare to finance. The book also addresses ethical considerations, challenges, and future prospects of LLMs, providing readers with a holistic view of this rapidly evolving field.

Supplementary Resources

As a supplementary resource, you can also refer to our Large Language Models (LLMs) System Design (2023 Edition) github repo for further insights on how to design deep learning systems for production.

http://bit.ly/llm-sys-design-2023

Contents

Chapter 1
LLM Foundations

1. Large Language Models (LLMs): An Intro

Unleashing the Power of Language

Large Language Models (LLMs) have emerged as transformative tools in the field of natural language processing, exhibiting remarkable capabilities in a broad spectrum of tasks. Trained on massive datasets of text and code, these models can generate human-quality text, translate languages seamlessly, answer complex questions comprehensively, and even write different kinds of creative content. This report delves into the fascinating world of LLMs, exploring their immense potential, deployment challenges, and cutting-edge solutions. In the following, we will start by an overview on language models in general and deep dive into large language models, their architecture, and applications.

1.1 Language Models

A Language Model (LM) is a computational model that predicts the probability of a sequence of words in a language. Another way to put it, a Language Model (LM) is an algorithm that uses statistical techniques to predict the likelihood of a given sequence of words appearing in a language, often used for generating or understanding human language text..

Modern LMs, especially those based on deep learning, can perform tasks like text generation, translation, summarization, and question answering by processing and producing human-like text.

Example

Let's consider a simple example to demonstrate how a Language Model (LM) works:

Imagine you're using an LM to predict the next word in the sentence:

"The cat sat on the ___."

The LM analyzes the sequence of words "The cat sat on the" and calculates the probability of what the next word could be based on its training. It might determine probabilities like this:

- "mat" - 60% probability
- "floor" - 20% probability
- "window" - 10% probability
- "table" - 5% probability
- Other words - 5% combined probability

Based on these probabilities, the LM would predict "mat" as the next word because it has the highest likelihood of being the correct continuation of the sentence, given typical patterns in English language usage.

This example simplifies the complex calculations and the vast amount of training data that real LMs use, but it captures the essence of how they predict text.

1.1.1 Language Models Overview

Modeling Techniques:

Early LMs were often based on statistical methods like n-grams, which predict the probability of a word based on the previous 'n-1' words.

Modern LMs predominantly use neural networks, particularly transformer-based architectures like BERT, GPT, and T5. These models capture complex patterns in language by processing sequences of text and learning contextual relationships between words.

Training Procedures:

Training a neural network-based LM involves feeding it large amounts of text data and adjusting the model's internal parameters to minimize the difference between its predictions and the actual text.

Autoregressive models like GPT are trained to predict the next word in a sequence, learning from each correct prediction.

Masked language models like BERT are trained using a technique where some words in the input text are randomly masked, and the model learns to predict these masked words based on their context.

Data Used

LMs are trained on extensive and diverse text corpora, often sourced from books, websites, articles, and other written material. The goal is to expose the model to as much of a language's variability as possible.

The quality and diversity of the training data significantly impact the model's performance and its ability to generalize across different types of text.

Challenges in Training:

Training advanced LMs requires substantial computational resources due to the size of the models and the volume of data processed.

Ensuring the training data is free from biases and is representative of a wide range of language use is also a significant challenge.

Fine-Tuning:

After initial training, LMs are often fine-tuned on specific tasks (like question answering or sentiment analysis) or specific datasets to enhance their performance in those areas.

In summary, modern LMs are sophisticated tools that learn the patterns and nuances of language through advanced neural network architectures and extensive training on diverse text

data. They've become foundational in natural language processing, enabling a wide range of applications in understanding and generating human language.

1.2 Large Language Models

TLDR: Large Language Model (LLM) is a fuzzy term that usually means a >1B parameter transformer-based model trained to predict the next token

Large Language Models (LLMs) are advanced, high-capacity neural network models specifically designed to understand, generate, and interact with human language at a sophisticated level.

These models, exemplified by the likes of GPT-3 and BERT, are trained on vast datasets comprising a wide array of textual sources, enabling them to grasp the nuances, context, and intricacies of language.

LLMs stand out for their deep understanding of syntax and semantics, allowing them to perform a variety of complex language tasks like text generation, translation, summarization, and question answering with remarkable proficiency.

Their size, measured in billions of parameters, gives them an unprecedented ability to generate coherent and contextually relevant text, making them powerful tools for a wide range of applications in technology, business, and research.

However, their large size also poses challenges in terms of computational resources, potential biases in training data, and ethical considerations in their deployment and use.

LLMs unlock LUIs (Language User Interfaces)

LLMs are key enablers of Language User Interfaces (LUIs), a form of user interface that allows people to interact with technology using natural language.

LLMs, with their advanced understanding and generation of human language, enable devices and applications to understand, interpret, and respond to user commands, questions, or inputs in natural, conversational language.

This advancement is transforming how users engage with technology, making it more intuitive and accessible by allowing communication in the user's own words rather than through structured commands or specific interface actions.

Language modeling is AI-Complete

The task of perfectly modeling human language is as complex as achieving Artificial General Intelligence (AGI). This implies that to fully and accurately model all aspects of human language, an AI would need to possess a broad, human-like understanding of the world, including context, culture, and abstract concepts, which is a hallmark of AGI. Essentially, solving language modeling completely would require solving many of the most challenging problems in AI.

We're building products finally! (Good news!)

Example products

- OpenAI's ChatGPT: a chatbot application powered by a variant of the GPT LLM. It specializes in generating human-like text responses in a conversational format.
- Replika: This is an AI companion chatbot that uses LLMs to engage users in conversation, learn from interactions, and provide a personalized experience.
- Duolingo: The language learning platform has experimented with GPT-3 to enhance its chatbot feature, making the conversations more natural and responsive, aiding language practice.
- Algolia's Answers: Utilizing LLMs, Algolia offers a feature called Answers, which provides highly relevant search results for queries on websites and apps by understanding the intent behind search queries.
- Jasper (formerly Jarvis): This is a content creation tool that uses AI to help write blog posts, social media content, and marketing copy, leveraging the power of LLMs to generate coherent and contextually appropriate text.
- YouChat by You.com: It's a search engine that uses LLMs to understand and answer user queries conversationally, providing direct answers and insights instead of just links.
- Adept AI: This startup is working on leveraging LLMs for automating office tasks like drafting emails or creating PowerPoint presentations by understanding user commands and context.
- Healx: In the healthcare sector, Healx uses AI, potentially including LLMs, for drug discovery, particularly for rare diseases, by analyzing medical research and data.
- Suki.AI: Suki is an AI-powered, voice-enabled digital assistant for doctors, helping them with tasks like note-taking and accessing patient data, leveraging LLMs to understand and process medical terminology and context.
- Legal Robot: This company uses AI, including LLMs, to help users understand legal language in contracts and documents, making it more accessible and comprehensible.
- Elevate Security: Focused on cybersecurity, they use AI to analyze and predict employee behavior that might pose security risks, potentially using LLMs to understand and process relevant data and communications.

LLM Capabilities and Potential Applications

- **Generating Human-Quality Text:** LLMs excel at creating text with remarkable fluency and coherence, imitating various writing styles and formats. This opens doors for applications like personalized content creation, generating marketing materials, and even writing creative fiction.
- **Translation and Language Understanding:** LLMs break down language barriers by translating between languages accurately and capturing the nuances of different

cultures. This paves the way for global communication, accessibility, and cross-cultural collaboration.

- **Question Answering and Knowledge Extraction:** LLMs can answer complex questions and extract valuable insights from vast amounts of information. This makes them invaluable for research, education, and customer service, offering efficient and informative solutions.
- **Creative Content Generation:** From writing poems and scripts to composing music and generating code, LLMs can unleash human creativity in new ways. This opens doors for artistic exploration, product development, and personalized experiences.

In the upcoming section, we delve deeper into the architecture and training processes of **ChatGPT**. Grasping these concepts is crucial for a comprehensive understanding of Large Language Models (LLMs) and their functioning. While this detailed knowledge is not mandatory for everyday users of ChatGPT or similar LLMs, it becomes invaluable for those looking to develop bespoke LLM solutions or preparing for job interviews in the LLM field.

2. ChatGPT and its Architecture

What is ChatGPT?

As mentioned earlier, ChatGPT is a chatbot application powered by a variant of the GPT (Generative Pretrained Transformer) LLM. It specializes in generating human-like text responses in a conversational format, leveraging the advanced language understanding and generation capabilities of LLMs to interact with users, answer questions, and provide information across a wide range of topics. ChatGPT is fine-tuned to offer coherent, contextually relevant, and engaging dialogue, making it a versatile tool for various conversational AI applications.

What is ChatGPT's Architecture?

The architecture of ChatGPT fundamentally utilizes **Transformer** blocks. Hence in the following, we begin by explaining the transformer architecture, progressing to the **GPT** architecture, and ultimately detailing the specific architecture of **ChatGPT**.

2.1 Transformer

The Transformer architecture is a revolutionary neural network architecture introduced in the paper "Attention Is All You Need" by Vaswani et al. It has had a profound impact on various natural language processing (NLP) and sequence-to-sequence tasks, achieving state-of-the-art results in machine translation, language modeling, text generation, and more. The Transformer is known for its ability to capture long-range dependencies in sequences efficiently and its parallelizability, making it highly scalable.

Transformer vs RNNs

Transformers and Recurrent Neural Networks (RNNs) are both NN architectures used in NLP and other tasks that require sequential processing, but they have distinct characteristics and operational mechanisms:

Architecture:

RNNs: Process sequences sequentially, updating a hidden state at each step.
Transformers: Use attention mechanisms to process sequences simultaneously, effectively capturing relationships between distant elements.
Parallelization:

RNNs: Sequential nature limits parallelization, leading to longer training times.
Transformers: Enable parallel processing of sequence elements, speeding up training and inference.
Handling Long-Range Dependencies:

RNNs: Struggle with long-range dependencies due to vanishing gradients.
Transformers: Excel at handling long-range dependencies through attention mechanisms.
Memory and Computational Requirements:

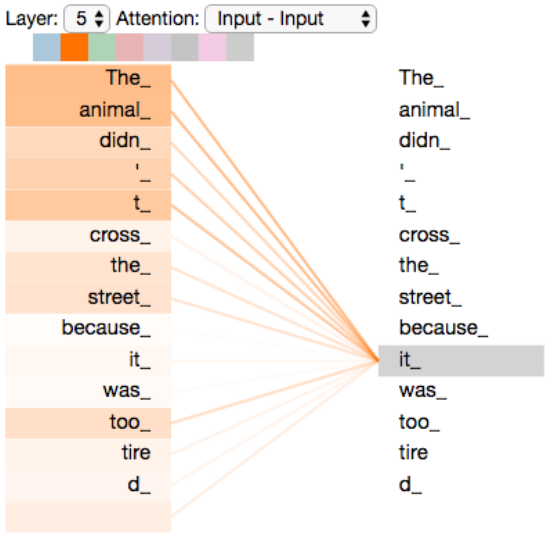
RNNs: Lower memory and computational requirements.
Transformers: Higher memory and computational demands due to their complex attention mechanism.
Use Cases:

RNNs: Effective in tasks like speech recognition and text generation.
Transformers: Preferred for complex NLP tasks like machine translation, text summarization, and question answering.

Attention Mechanism

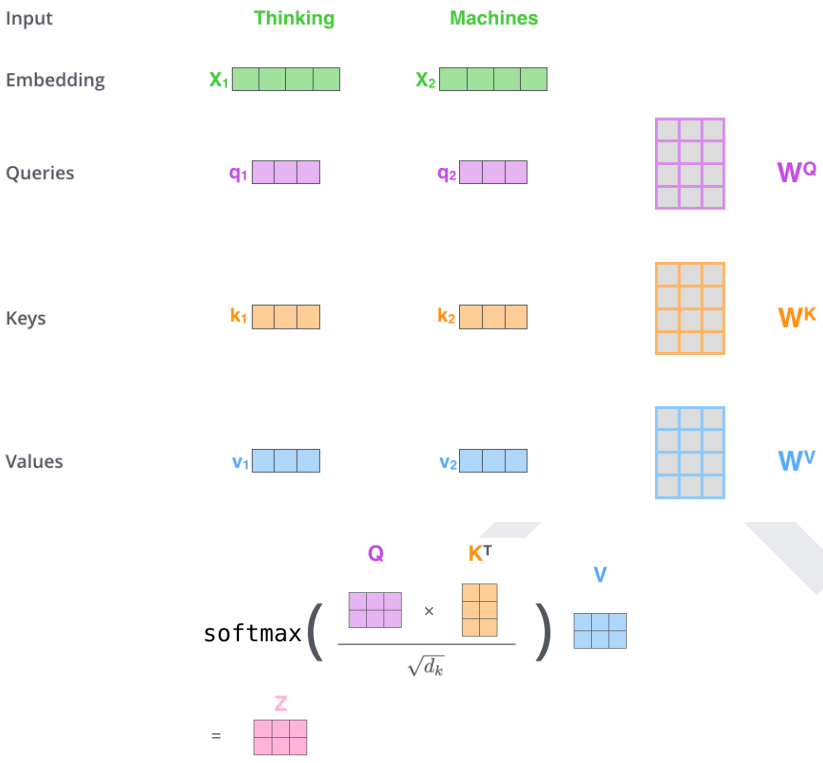
"Attention Is All You Need!" - Ashish Vaswani et al

Self-Attention Mechanism:



The self-attention mechanism is the cornerstone of the Transformer architecture. It enables the model to weigh the relevance of each word/token in the input sequence when making predictions for a specific word/token. Here's how it works:

- For each word/token in the input sequence, three vectors are computed:
    - Query vector: This represents the word/token for which we want to determine the importance.
    - Key vector: These vectors represent the words/tokens against which we compare the query vector.
    - Value vector: These vectors contain information that is used to update the query word/token.
- To compute the attention scores, the dot product between the query and key vectors is taken. The result is then scaled, often by the square root of the dimension of the key vectors, to ensure stable gradients.
- The scaled dot products are passed through a softmax function to produce attention weights that sum to 1. These weights represent how much each word/token in the sequence contributes to the prediction for the query word/token.
- Finally, the attention weights are used to compute a weighted sum of the value vectors, generating the final output for the query word/token.



Transformer Architecture

Here's an explanation of the key components of the Transformer architecture:

Multi-Head Attention:

To capture different types of relationships and patterns in the input sequence, the Transformer employs multiple "attention heads" in parallel. Each attention head learns different patterns and provides a unique perspective on the data. The outputs of these attention heads are concatenated and linearly transformed to form the final multi-head attention output.

Positional Encoding

Since the Transformer does not inherently account for the order of words/tokens, positional encodings are added to the input embeddings. These encodings provide information about the position of each word/token in the sequence and are summed with the embeddings.

One common method for positional encoding is to use sinusoidal functions, which have different frequencies for different positions. This ensures that the model can distinguish between the positions of words/tokens effectively.

Stacked Encoder-Decoder Architecture (in Sequence-to-Sequence Tasks):

In tasks like machine translation, the Transformer employs an encoder-decoder architecture:

- The encoder processes the input sequence using self-attention layers and feedforward neural networks to capture contextual information about the input.
- The decoder, on the other hand, uses masked self-attention to prevent future positions from being attended to and generates the output sequence one word/token at a time.
- The encoder's final hidden states are used as context information for the decoder. This helps the decoder generate output words/tokens conditioned on the input sequence.

Feedforward Neural Networks (FFNs):

After the attention layers, both the encoder and decoder include feedforward neural networks (FFNs). These networks consist of fully connected layers and ReLU activation functions. FFNs enable the model to capture complex, non-linear transformations and interactions between features.

Layer Normalization and Residual Connections:

Layer normalization is applied after each sub-layer (e.g., attention layer, feedforward layer), which helps stabilize and speed up training. Residual connections, inspired by the residual network (ResNet) architecture, allow gradients to flow directly through the network. They are implemented as skip connections that bypass a sub-layer and are added to its output.

Stacked Layers:

Transformers are typically composed of multiple layers stacked on top of each other. The stack of layers enables the model to capture increasingly abstract and high-level patterns as it progresses through the network.

The following figure, inspired by “The Illustrated Transformer” blog by Jay Alamar, illustrates the whole Transformer architecture in a simplified format:

Putting it all together

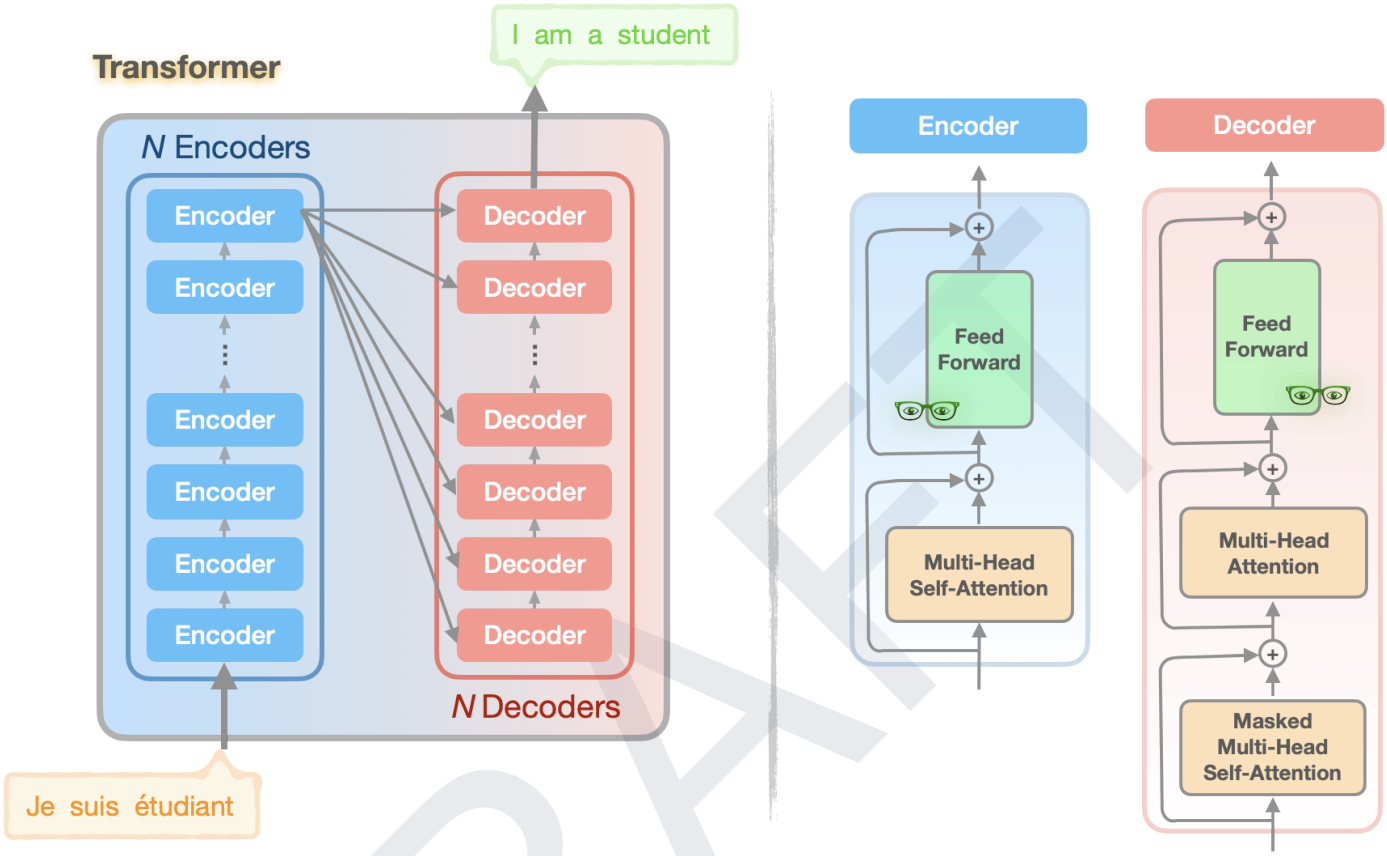


Fig. 1.X: A simplified summary of the standard Transformer model [Image inspired by 'The Illustrated Transformer']

Loss Functions

Transformers utilize a variety of loss functions depending on the specific task and desired outcome. Here are some of the commonly used ones:

Cross-Entropy Loss:

- This is the most common loss function for language modeling tasks like predicting the next word in a sequence. It measures the difference between the model's predicted probability distribution over the vocabulary and the true distribution (one-hot encoding for the actual next word).
- It encourages the model to assign high probabilities to the correct next words and low probabilities to incorrect ones.

Masked Language Modeling (MLM) Loss:

- This loss function is used for tasks like filling in masked words in a sentence. It works by masking certain words in the input and then predicting the masked words based on the surrounding context.
- This helps the model learn long-range dependencies and improve its ability to understand the relationships between words in a sentence.

Perplexity:

- This is not technically a loss function but a metric related to cross-entropy loss that measures the model's ability to predict the next word. A lower perplexity indicates a better model performance.

Other Loss Functions:

- Sentence-level losses: These are used for tasks like evaluating the semantic similarity of two sentences. Examples include BLEU score, ROUGE, and BERTScore.
- Ranking losses: These are used for tasks like ranking different outputs based on their relevance to a query. Examples include Triplet Loss and Margin Ranking Loss.
- Discriminative losses: These are used for tasks like adversarial training, where the model is trained to fool a discriminator while the discriminator is trained to distinguish between real and generated data. Examples include Generative Adversarial Networks (GAN) loss and Wasserstein GAN (WGAN) loss.

The choice of the appropriate loss function depends on the specific task and the desired model behavior. Some factors to consider include:

- Task requirements: What is the desired output of the model? Is it accuracy, fluency, or something else?
- Training data: What are the characteristics of the training data? Is it noisy, limited, or diverse?
- Model architecture: What are the strengths and weaknesses of the chosen Transformer architecture?

By carefully choosing and tuning the loss function, developers can optimize Transformer models for specific tasks and achieve desired performance.

I hope this information provides a good overview of the loss functions used in Transformer architectures. If you have any further questions about specific loss functions or their applications, feel free to ask!

Challenges with Transformers:

The basic transformer architecture, while powerful, faces several key challenges:

1. Computational Cost: Transformers require significant computational resources, especially for training. Their fully connected nature leads to high memory and processing demands, particularly with large input sequences.

2. Scalability: Handling very long sequences is a challenge due to *quadratic complexity with respect to sequence length*. This makes it difficult to scale transformers for tasks with extremely long text inputs.

3. Data Efficiency: Transformers often require large amounts of training data to perform well. This can be a limitation in scenarios where large, labeled datasets are not available.

4. Overfitting: Due to their high capacity (number of parameters), transformers can overfit on smaller datasets, learning noise and specifics of the training data rather than generalizing well.

5. Interpretability: Understanding how transformers make decisions or what features they are focusing on can be challenging, which is a common issue in many complex neural network architectures.

6. Generalization: While transformers perform exceptionally well on the tasks they are trained for, transferring that learning to significantly different tasks or domains can be challenging without substantial retraining or fine-tuning.

These challenges have led to ongoing research and development in the field, aiming to create more efficient, scalable, and versatile transformer models.

Variants and Improvements:

Since its introduction, the Transformer architecture has undergone numerous variants and improvements. Variants like BERT, GPT, and T5 have adapted the Transformer architecture for various NLP tasks. These models have different pre-training and fine-tuning strategies, making them versatile and effective across a wide range of applications.

The Transformer architecture has made it possible to process and generate sequences of data with remarkable accuracy and efficiency. It has been the foundation for several NLP models and beyond and continues to advance the field of deep learning for sequence-based tasks.

The original transformer architecture, introduced in the paper "Attention Is All You Need" by Vaswani et al., revolutionized many aspects of NLP and AI. However, it does have some drawbacks, which have led to various improvements over the years. Here's an overview of the drawbacks and notable improvements, including those relevant to models like GPT-4 and Gemini:

Drawbacks of the Original Transformer:

Quadratic Complexity: The self-attention mechanism in transformers has a quadratic computational complexity with respect to the sequence length, making it resource-intensive for long sequences.

Memory Intensive: Transformers require substantial memory, which can be a limiting factor, especially when dealing with longer sequences or larger models.

Parallelization Limitations: While transformers are more parallelizable than RNNs, the self-attention mechanism still poses challenges for efficient parallelization, impacting training and inference times.

Improvements to Transformer Architecture:

Efficient Attention Mechanisms:

- Linear Transformers: These use kernels or approximations to reduce the complexity of the self-attention mechanism from quadratic to linear, making them more efficient for processing long sequences.
- Sparse Transformers: Introduce sparsity into the attention mechanism, allowing the model to focus on a subset of the input sequence, thereby reducing computational requirements.

Model Parallelism and Sharding:

- Developments in model parallelism, where different parts of the transformer model are processed on different hardware units, have improved training efficiency.
- Sharding techniques, where the model parameters are distributed across multiple devices, help in managing large-scale models like GPT-4.

Adaptive Computation:

- Techniques like mixture-of-experts, where different parts of the network specialize in different types of information, allow for more efficient computation by activating only relevant parts of the model for a given input.

Architectural Variants for Specific Tasks:

- Models like GPT-4 and Gemini incorporate various architectural tweaks and improvements tailored to their specific application domains, such as optimizing the number of layers, attention heads, and embedding sizes for better performance and efficiency.

These improvements have significantly enhanced the capabilities and efficiency of transformer models, enabling the development of more advanced and scalable LLMs. They address the core limitations of the original design, allowing for better handling of longer sequences, more efficient memory usage, and faster training and inference times.

The memory-intensive nature of transformers, particularly in the context of Large Language Models (LLMs), is a significant challenge. This issue arises due to several factors inherent to the transformer architecture:

Quadratic Scaling with Sequence Length:

The self-attention mechanism in transformers computes relationships between every pair of tokens in the input sequence. As a result, the memory required scales quadratically with the sequence length. For very long sequences, this can lead to extremely high memory usage, making it challenging to process them effectively on standard hardware.

Large Number of Parameters:

LLMs like GPT-4 have billions of parameters. Storing these parameters, along with the intermediate computations required during training and inference, demands substantial memory. This is especially true during training, where additional memory is required for gradients and optimizer states.

Activation Maps:

During the forward and backward passes in the training process, transformers need to store activation maps (intermediate outputs of each layer). These activation maps are used to compute gradients during backpropagation. As the depth (number of layers) and width (size of each layer) of the model increase, so does the memory required to store these activations.

Batch Processing:

Training and inference often involve processing multiple instances of data (batch processing) simultaneously for efficiency. This multiplies the memory requirement, as each instance in the batch requires its own set of activations and intermediate calculations.

To address these challenges, various strategies and optimizations are employed, such as:

Gradient Checkpointing: Saving only a subset of activations and recomputing others as needed during backpropagation.

Model Parallelism: Distributing different parts of the model across multiple GPUs or TPUs to share the memory load.

Memory-Efficient Attention Mechanisms: Using techniques like linearized attention to reduce the memory footprint of the attention mechanism.

In the following, we will briefly review one of the most popular variants of the transformer, GPT, which is used as the LLM in ChatGPT.

2.2 GPT

TLDR: GPT (Generative Pre-trained Transformer) is a decoder-only NLP model with a Transformer architecture, renowned for its autoregressive text generation capabilities. The number of decoder layers and the size of the model vary across different versions of GPT, with larger models having more layers and parameters to capture increasingly complex language patterns.

GPT, which stands for "Generative Pre-trained Transformer," is a groundbreaking series of NLP models developed by OpenAI. What distinguishes GPT from other NLP models is its **decoder-only** architecture, which is based on the Transformer architecture. Here's a comprehensive explanation, including architectural details and model sizes:

Transformer Architecture (Decoder-Only)

GPT is built upon the Transformer architecture The Transformer architecture marked a significant advancement in NLP with its self-attention mechanisms and parallelizable design.

Decoder-Only Design:

One of the distinctive features of GPT is its decoder-only design. In the original Transformer architecture, both encoder and decoder components are used. The encoder processes input data, while the decoder generates output sequences. However, GPT focuses exclusively on text generation and language modeling, making it a decoder-only model.

Autoregressive Text Generation

GPT's primary strength lies in autoregressive text generation. In this mode, GPT generates text sequentially, one word at a time, in a left-to-right manner. It uses the previously generated words as context to predict the next word. This autoregressive approach allows GPT to produce coherent and contextually relevant text.

Stacked Decoder Layers

GPT models consist of a stack of identical decoder layers. Each decoder layer includes self-attention mechanisms and feedforward neural networks. The depth of the stack and the size of the model vary across different GPT versions:

- GPT-1): Typically has 12 decoder layers.
- GPT-2: Has 24 decoder layers.
- GPT-3: One of the largest models, with 175 billion parameters, but the specific number of decoder layers is undisclosed. It includes smaller versions with fewer parameters.

Pre-training and Fine-Tuning

GPT models undergo a two-phase training process:

- **Pre-training:** During this phase, GPT is trained on a massive corpus of text data, typically containing a substantial portion of internet text. The primary objective of pre-training is to predict the next word in a sentence, encouraging the model to learn rich semantic and contextual information from the data.
- **Fine-Tuning:** After pre-training, GPT can be fine-tuned for specific downstream NLP tasks. Fine-tuning involves adding task-specific layers on top of the pre-trained GPT model and training it on task-specific data. Fine-tuning adapts the general language

understanding capabilities of GPT to perform tasks such as text classification, summarization, question-answering, and more.

Applications

GPT's decoder-only design and architecture make it well-suited for a wide range of text generation tasks, including:

- Creative writing (e.g., generating stories, poetry, and content).
- Text summarization.
- Language translation.
- Chatbots and virtual assistants.
- Code generation.
- Text completion.
- Content generation for marketing and advertising.

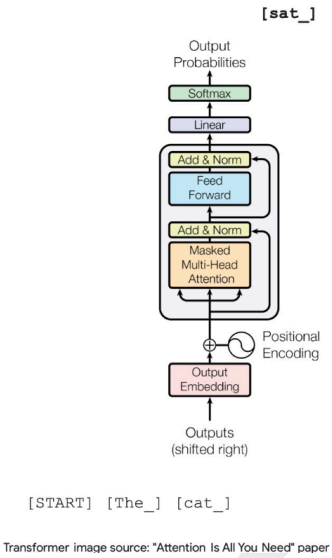
Ethical Considerations

The success of GPT models, especially larger versions like GPT-3, has raised ethical concerns related to biases in generated text and potential misuse. Addressing these concerns and ensuring responsible AI deployment are ongoing challenges in the development and use of GPT models.

2.2.1 GPT Progression

GPT / GPT-2 (2019)

- Generative Pre-trained Transformer
- Decoder-only (uses masked self-attention)
- Largest model is 1.5B



Transformer image source: "Attention Is All You Need" paper

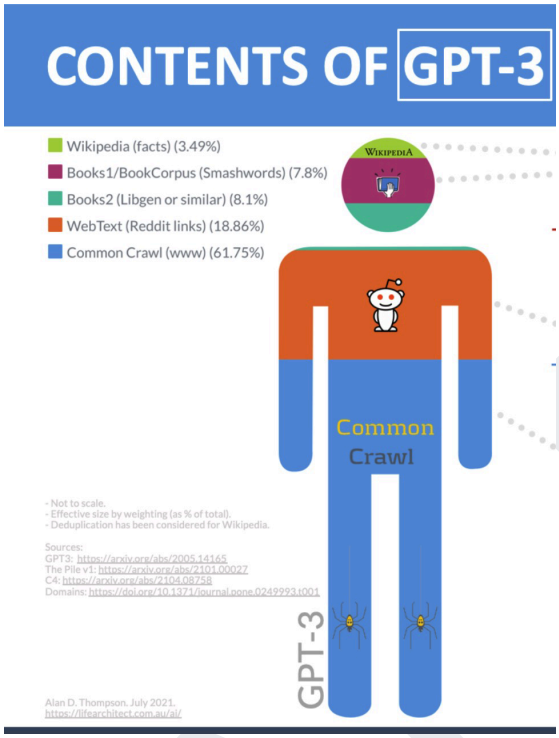
GPT-2 Training Data

- Found that Common Crawl has major data quality Issues
- Formed the WebText dataset
    - scraped all outbound links (45M) from Reddit which received at least 3 karma
- After de-duplication and some heuristic filtering, left with 8M documents for a total of 40GB of text

GPT-3 (2020)

- Just like GPT-2, but 100x larger (175B params)
- Exhibited unprecedented few-shot and zero-shot learning

GPT-3 Training Data



Computation: about 200 million petaFLOP for GPT-3, 1 million for GPT-2

GPT-3.5 or InstructGPT (202X)

- Had humans rank different GPT-3 outputs, and used RL to further fine-tune the model
- Much better at following instructions

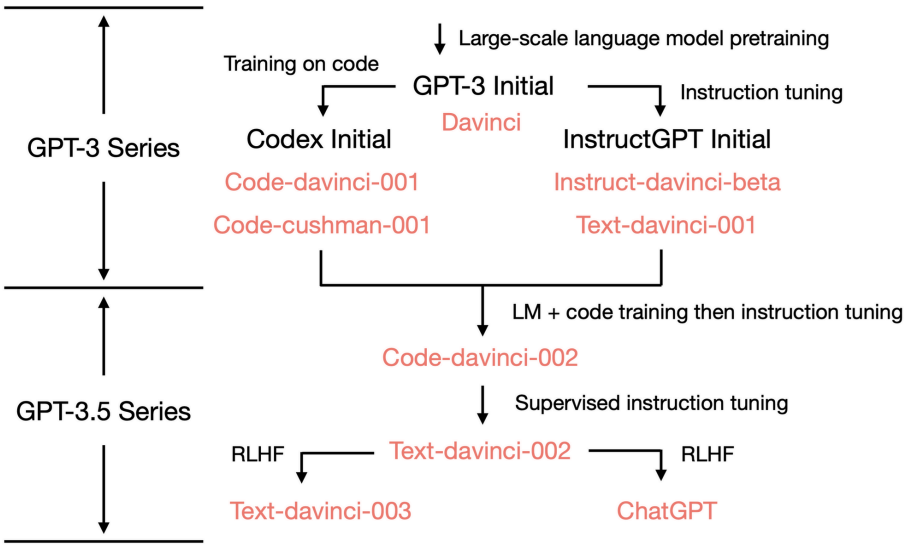
GPT-4 (2023)

GPT-4 is a Transformer style model pretrained to predict the next token in a document, using both publicly available data (such as internet data) and data licsened from third party providers.

The model was then fine tuned using Reinforcement Learning from Human Feedback (RLHF).

No further details about architecture (model size), hardware, training compute, dataset construction, training method, etc available due to competitive landscape and safety implications of LLMs.

- One lesson: Stack more layers



2.3. ChatGPT

ChatGPT is a variant of the GPT LLM developed by OpenAI, designed and fine-tuned specifically for conversational applications. It shares many similarities with its predecessor, GPT-3, but it has been fine-tuned and adapted for improved performance in generating human-like text in conversational contexts.

2.3.1 ChatGPT's Architecture

The architecture of ChatGPT is built upon the foundation of the GPT model, with specific adaptations for conversational tasks. Here are the key aspects of its architecture:

- **Transformer-Based Model:** At its core, ChatGPT uses the transformer architecture, renowned for its effectiveness in handling sequential data and capturing context in language. This architecture comprises multiple layers of self-attention and feed-forward neural networks.
- **Large Number of Parameters:** Similar to other GPT models, ChatGPT has a large number of parameters (the exact number depends on the specific version, like GPT-3.5, GPT-4, etc.). These parameters allow the model to capture and generate a wide range of linguistic patterns and responses.
- **Context Window:** ChatGPT is designed to handle a significant context window, allowing it to refer to and incorporate previous parts of the conversation. This is crucial for maintaining coherence over longer interactions.
- **Fine-Tuning for Dialogue:** While based on the standard GPT architecture, ChatGPT is fine-tuned on conversational data. This fine-tuning process tailors the model's responses to be more suitable for dialogues, improving its performance in interactive scenarios.

- **Reinforcement Learning from Human Feedback:** An additional layer of training through RLHF (Reinforcement Learning from Human Feedback) refines its responses further, based on evaluators' rankings of model outputs, making the responses more aligned with human conversational norms and preferences.
- **Safety and Ethical Considerations:** The architecture is also designed to incorporate mechanisms to handle safety and ethical issues, ensuring that the responses generated are appropriate and non-offensive.

In summary, ChatGPT's architecture combines the advanced capabilities of the GPT transformer model with specialized fine-tuning and training to excel in conversational applications, ensuring relevance, coherence, and appropriateness in its interactions.

2.3.2 ChatGPT's Training

ChatGPT's training process and data handling involve several key aspects that contribute to its performance and capabilities in conversational contexts. There are **three main stages** of training ChatGPT's model:

1. Pre-Training Phase

ChatGPT, like other GPT models, undergoes extensive pre-training on a diverse range of internet text. This pre-training involves learning to predict the next word in a sequence, helping the model to understand language patterns, grammar, and context.

The goal here is to learn a broad understanding of language, context, and patterns.

Pre-training does not involve task-specific data or instructions.

2. Fine-Tuning Phase

After pre-training, ChatGPT is fine-tuned on conversational data. This step adjusts the model to be more effective in dialogue-specific tasks. The fine-tuning process involves using datasets specifically designed to mimic conversational structures and interactions.

Note: Few-shot vs Zero-shot:

- At the time of GPT-3 (2020), the mindset was mostly few-shot - e.g. text completion
- By the time of ChatGPT (2022), the mindset was all zero-shot - e.g. instruction-following

- **Supervised Fine-Tuning:** This dataset consists of prompts and their corresponding responses, helping the model learn appropriate replies in a conversational context.
    - Very little text in the original GPT-3 dataset is of the zero-shot form.
    - To improve performance on zero-shot inputs, fine-tuned on a smaller high-quality dataset of instructions completions (Sourced from thousands of contractors)

- **Instruction Tuning:** This is a specialized form of fine-tuning where the model is trained to follow and respond to natural language instructions more accurately. It helps the model in understanding and executing a wide range of user commands.

3. Reinforcement Learning from Human Feedback (RLHF) Phase

RLHF is a crucial phase in ChatGPT's training. It involves human trainers providing the model with feedback on its responses. Based on this feedback, the model is further refined using reinforcement learning techniques to align its responses more closely with human preferences and expectations in a conversation.

What is RLHF?

RLHF is a method where a language model is refined based on feedback from human evaluators. The process involves humans reviewing and rating the model's responses, and then using these ratings to train the model further.

Models Used in RLHF

The models involved in RLHF are typically already trained LLMs like GPT-3. These models have gone through initial pre-training and fine-tuning stages and are capable of generating coherent text.

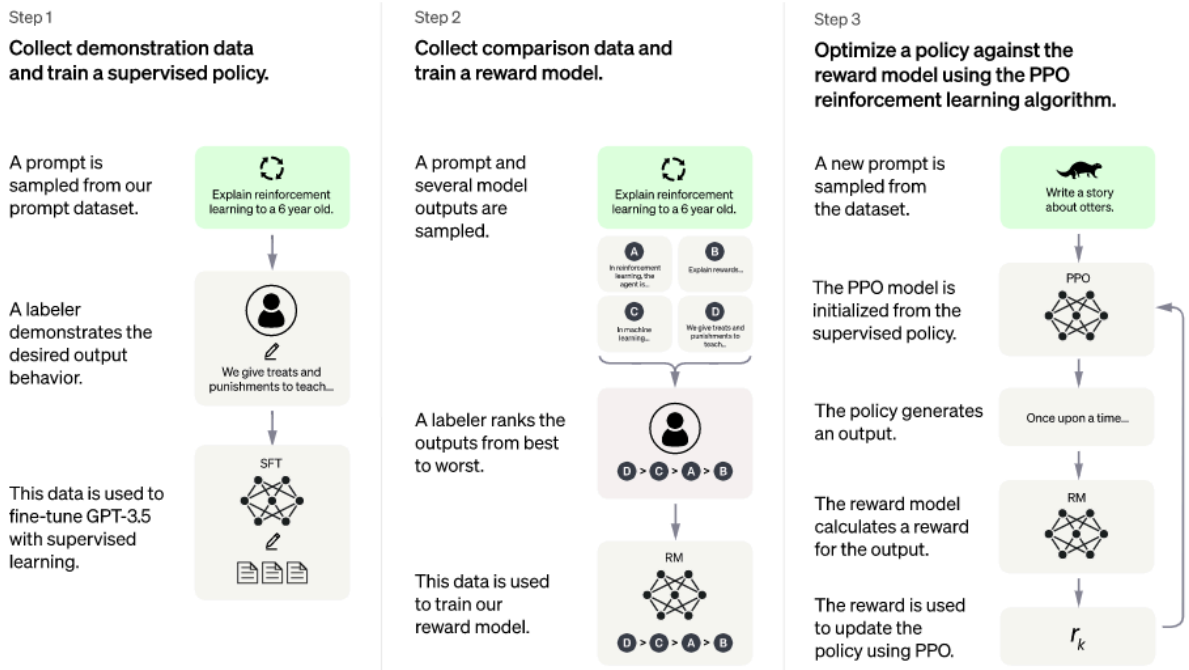
Architecture

The architecture in RLHF doesn't necessarily change from the base LLM; instead, the focus is on the training procedure. The LLM continues to use its transformer-based architecture.

A **reward model** is often developed, which is trained to predict the quality of responses based on human feedback. This reward model then guides the reinforcement learning process.

Training Process in RLHF

- **Human Evaluation:** Trainers review the model-generated responses in various scenarios and provide ratings or preferences.
- **Reward Modeling:** A separate model is trained to predict these human ratings, learning to assess the quality of responses.
- **Policy Training:** The LLM is then further trained using reinforcement learning, where it's encouraged to generate responses that are predicted to be highly rated by the reward model.



A brief background on Reinforcement Learning

- Basic Concept: In RL, an **agent** learns to make decisions by performing **actions** in an **environment** to achieve a goal. The agent receives **rewards** or penalties based on its actions, learning over time to *maximize cumulative rewards*.
- Applicability to LLMs: For LLMs in RLHF, the 'environment' is typically a simulation of conversational scenarios, and the 'actions' are the generation of text responses. The goal is to generate responses that align with desired outcomes (e.g., being informative, coherent, and contextually appropriate).
- **Reward Signal**: In RLHF, the reward signal is often derived from the evaluations provided by the reward model, which judges the quality of the language model's responses based on human feedback.
- **Policy Training**: a policy is a strategy or a set of rules that an agent follows to decide on actions based on the current state of the environment. Policy training involves iteratively adjusting the model's parameters to learn a strategy that maximizes cumulative rewards based on feedback from its interactions with the environment.

Reward Modeling

The reward model used for training of LLMs typically employs a transformer-based NN architecture, leveraging the strengths of these models in language understanding but fine-tuned for the specific task of evaluating and scoring language model responses.

Policy Training:

- Defining Policy: In the context of RLHF for LLMs, the policy defines how the language model decides to generate responses based on the given input and its current state. The policy is typically represented by a NN model (often a transformer-based architecture). This choice is due to the transformer's effectiveness in handling complex patterns in language and its ability to generate coherent and contextually appropriate responses in conversational AI applications.
- Training the Policy: Policy training involves adjusting the parameters of the language model so that it maximizes the expected rewards. This is done through trial and error, where the model explores different ways of responding and learns from the feedback (rewards or penalties).

Proximal Policy Optimization (PPO):

- Algorithm Overview: PPO is a popular RL algorithm used for **Training Policies**. It's known for its stability and reliability in training complex policies, like those needed for LLMs.
- Key Features: PPO tries to take the biggest possible step to improve the policy without stepping so far that it causes training instability. It does this by using a clipped objective function, which limits the size of the policy update at each step.
- The objective function is designed to improve the policy while keeping the updates relatively stable and conservative. The key component of the PPO objective function is the clipped surrogate objective, which is defined as follows:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right]$$

(For more info ask ChatGPT 🙂).

- Use in LLMs: In the case of LLMs, PPO can be used to fine-tune the model on the task of generating appropriate responses. The algorithm helps in efficiently finding the policy that yields the highest reward as judged by the reward model, while avoiding drastic changes that could lead to unstable or erratic behavior.

RLHF in LLMs (ChatGPT)

- For ChatGPT, RLHF is used to align the model's responses with what is considered *appropriate, informative, and engaging* by human standards.
- It involves careful consideration of ethical guidelines, avoiding biases, and ensuring safety in responses.
- Iterative feedback and training cycles are used to continuously improve the model's performance and alignment with desired conversational behaviors.
- ChatML format: messages from system, assistant, user roles

```
# Note: you need to be using OpenAI Python v0.27.0 for the code below to work
import openai

openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the Worl
        {"role": "user", "content": "Where was it played?"}
    ]
)
```

Challenges and Considerations

- One of the key challenges in RLHF is ensuring that the human feedback is unbiased and representative.
- Balancing the model's creativity and adherence to safe and ethical response guidelines is also a crucial part of the process.

4. More on ChatGPT Training

Data Quality and Diversity

The quality and diversity of training data are vital. ChatGPT's training involves a broad spectrum of text sources to ensure it can handle a wide variety of topics, languages, and conversational styles.

Handling Biases and Sensitivities

An important part of training involves minimizing biases and ensuring sensitivity to various topics. This is done through careful selection and curation of training data, along with fine-tuning strategies that aim to reduce unwanted biases in model outputs.

Continual Learning and Updating

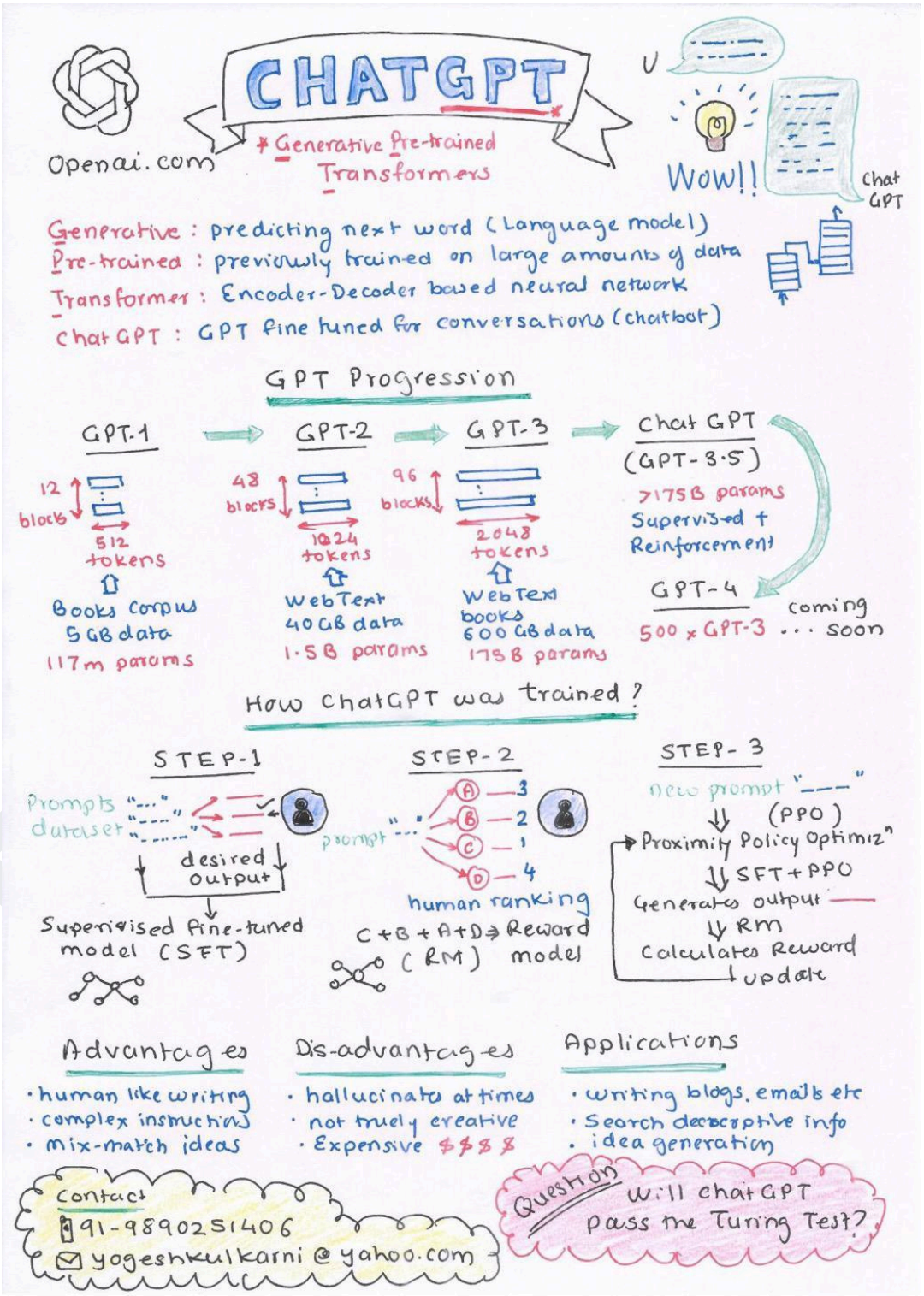
Given the dynamic nature of language and conversations, ChatGPT's training is not a one-time process. The model undergoes periodic updates to stay current with new information, trends, and language usage.

Safety and Ethical Considerations

Safety measures are integrated into the training process to ensure that the model avoids generating harmful or inappropriate content. This involves both technical strategies and guidelines implemented during the training and feedback phases.

Through these training methods and considerations, ChatGPT is designed to be a robust, versatile, and contextually aware conversational agent, capable of engaging users in meaningful and coherent interactions.

The following diagram [REF] summarizes some of the main aspects of ChatGPT very well.



2.3.3 ChatGPT Competitors

Google AI's Bard

- Concept and Functionality: Bard is an experimental AI conversational model developed by Google. It is designed to integrate seamlessly with the internet, pulling in real-time information to provide accurate and up-to-date responses.
- Engagement and Information Delivery: Aims to deliver information in an engaging, conversational manner, utilizing Google's vast search data and capabilities.

Comparing Bard with ChatGPT

- Architecture: While Bard is built to harness live web data, ChatGPT, developed by OpenAI, is based on the GPT architecture and relies on a static dataset up to its last training cut-off.
- Real-Time Data Access: Bard's key advantage lies in its ability to access and incorporate current information from the web, offering more up-to-date responses compared to ChatGPT.
- Knowledge Base: ChatGPT's responses, although extensive and varied, are limited by the scope of its training data and lack real-time updates.
- Interactivity and Engagement: Both models aim for high-quality, engaging interactions, but Bard's access to real-time data potentially allows for more current and contextually relevant conversations.

Key Distinction:

The primary difference between Google AI's Bard and OpenAI's ChatGPT is Bard's integration with live internet data, offering more current and dynamic responses, as opposed to ChatGPT's reliance on pre-trained knowledge.

Chapter 2

LLM System Design Overview

[To-Do]: Add more context on LLM System design intro

1. LLMs in Production

1.1 LLMs in Production: Challenges and Solutions

1.1.1 Challenges with LLMs in Production

Category 1: Model Performance and Generalizability

- **Precise Task Execution:** This includes challenges like ensuring the LLM accurately understands and executes the intended task, even with complex or ambiguous prompts.
- **Limited Knowledge Scope:** This refers to the LLM's ability to access and utilize relevant information beyond its training data.
- **Overfitting:** This occurs when the LLM memorizes the training data instead of learning generalizable patterns, leading to poor performance on unseen data.

Category 2: Infrastructure and Deployment

- **Scalability and Reliability in Production:** This involves ensuring the LLM can handle real-world workloads efficiently and consistently, without outages or performance degradation.
- **LLMOps and Development Practices:** This encompasses the tools and methodologies for effectively managing, monitoring, and improving LLMs in production environments.
- **Cost and Resource Efficiency:** This refers to optimizing the computational resources required to train and run LLMs, making them more accessible and cost-effective.

Category 3: Societal and Ethical Considerations

- **Explainability and Bias:** This highlights the need for transparent and accountable LLMs that avoid perpetuating harmful biases or producing misleading outputs.
- **Security and Privacy:** This addresses the vulnerabilities of LLMs to attacks and data breaches, requiring robust security measures and user privacy protection.
- **User Interface and Integration:** This emphasizes the importance of designing user-friendly interfaces for interacting with LLMs, particularly for non-technical users.

1.1.2 Solutions for LLMs in Production

Category 1: Model Performance and Generalizability

- **Precise Task Execution**
  - Prompt engineering: Carefully crafting prompts that clearly define the desired task and provide relevant context can improve LLM understanding and accuracy.
  - Task decomposition: Breaking down complex tasks into smaller, more manageable subtasks can improve LLM performance and enable more precise execution.
  - Fine-tuning: Adapting pre-trained LLMs to specific tasks through additional training on relevant data can enhance their performance in particular domains.
- **Limited Knowledge Scope:**
  - Augmenting with IR systems: Integrating LLMs with information retrieval (IR) systems allows them to access and process external knowledge bases, expanding their information scope.
  - Augmented LLMs: Developing LLMs with built-in knowledge retrieval capabilities can enable them to access and utilize relevant information without relying on external systems.
  - Tools for knowledge acquisition: Building tools and techniques for LLMs to actively learn and acquire new knowledge from various sources can improve their long-term knowledge scope.
- **Overfitting:**
  - Data augmentation: Artificially increasing the diversity and complexity of the training data can help the LLM learn generalizable patterns and avoid overfitting on specific examples.
  - Regularization techniques: Applying techniques like dropout and weight decay during training can encourage the LLM to rely less on memorized patterns and improve its generalization ability.
  - Curriculum learning: Gradually increasing the difficulty of training data as the LLM learns can force it to adapt and refine its understanding, reducing the risk of overfitting.

Category 2: Infrastructure and Deployment

- **Scalability and Reliability in Production:**
  - LLMOps: Implementing LLMOps practices like model monitoring, versioning, and deployment pipelines can ensure smooth and reliable operation in production environments.
  - Cloud-based platforms: Leveraging cloud platforms with scalable infrastructure and managed services can facilitate efficient deployment and scaling of LLMs.

    - Resource optimization techniques: Employing techniques like model quantization and efficient inference algorithms can reduce the computational resources required to run LLMs.
- **LLMOps and Development Practices:**
    - Test-driven development (TDD): Applying TDD principles to LLM development can help identify and address potential issues early on, leading to more robust and reliable models.
    - Continuous learning and adaptation: Implementing mechanisms for LLMs to continuously learn and adapt to new data and feedback can improve their performance and relevance over time.
    - Open-source collaboration: Sharing knowledge and best practices through open-source initiatives can benefit the entire LLM development community and accelerate progress.
- **Cost and Resource Efficiency:**
    - Model compression and pruning: Techniques like model compression and pruning can reduce the size and resource requirements of LLMs, making them more efficient to deploy and run.
    - Green AI initiatives: Implementing green AI practices like energy-efficient hardware and training methods can reduce the environmental impact of LLMs.
    - Cost-effective training approaches: Exploring alternative training approaches like federated learning or transfer learning can reduce the costs associated with training large LLMs.

Category 3: Societal and Ethical Considerations

- **Explainability and Bias:**
    - Explainable AI (XAI) techniques: Developing XAI techniques to provide insights into LLM reasoning and decision-making processes can improve transparency and address bias concerns.
    - Human-in-the-loop approaches: Integrating human oversight and feedback into LLM systems can help mitigate bias and ensure responsible use of these powerful tools.
    - Fairness-aware training and evaluation: Employing fairness-aware training methods and evaluation metrics can help identify and mitigate potential biases in LLMs.
- **Security and Privacy:**
    - Robust security measures: Implementing robust security measures like encryption and access control can protect LLMs from unauthorized access and data breaches.
    - Privacy-preserving techniques: Utilizing privacy-preserving techniques like differential privacy can help protect user data and privacy while still enabling LLMs to learn and perform effectively.
    - Transparency and user education: Educating users about potential risks and limitations of LLMs can help promote responsible use and build trust.
- **User Interface and Integration:**

- User-centered design principles: Applying user-centered design principles to LLM interfaces can ensure they are intuitive, accessible, and meet the needs of diverse users.
- Natural language interaction (NLI): Integrating natural language interaction capabilities into LLMs can enable more natural and user-friendly interactions.
- Context-aware interfaces: Developing LLMs with context-aware interfaces can personalize interactions and adapt to user needs and preferences.

Remember, these are just a few examples, and the most effective solutions will depend on the specific

1.1.1 More on Overfitting in LLMs

Overfitting in LLMs: Memorizing vs. Learning the Language

Large language models (LLMs) are incredibly powerful tools capable of generating human-quality text, translating languages, and answering complex questions. However, like any machine learning model, they are susceptible to overfitting, a phenomenon where the model memorizes the training data instead of learning the underlying patterns and relationships.

How Overfitting Manifests in LLMs?

- **Lack of generalization:** An overfitted LLM excels on the training data it has seen but performs poorly on unseen data. This can lead to nonsensical outputs or irrelevant responses when presented with novel prompts or questions.
- **Biased outputs:** Overfitting can amplify existing biases in the training data, leading to discriminatory or unfair language generation or responses.
- **Reduced creativity and originality:** An LLM that simply regurgitates memorized patterns loses its ability to generate creative and insightful text or solutions.

Solutions to Overfitting in LLMs:

- **Data Augmentation:** Introduce variations and perturbations into the training data to increase its diversity and challenge the model to learn the underlying patterns. Techniques like synonym replacement, back-translation, and random masking can be effective.
- **Regularization:** Apply techniques like dropout, weight decay, and early stopping to discourage the model from memorizing specific training examples and promote generalization.
- **Transfer Learning:** Leverage pre-trained models on large datasets to provide the LLM with a strong foundation of language understanding and reduce the need to memorize specific training data.

- **Curriculum Learning:** Gradually increase the difficulty of the training data as the LLM learns, forcing it to adapt and refine its understanding instead of relying on memorized patterns.
- **Meta-Learning:** Train the LLM to learn how to learn effectively, enabling it to adapt to different tasks and data distributions and avoid overfitting.
- **Ensemble Learning:** Combine multiple LLMs with different strengths and weaknesses to create a more robust and resilient model less susceptible to overfitting on any individual training data set.

Monitoring and Addressing Overfitting

- Track training and validation losses: Monitor the difference between how the LLM performs on the training data and the validation data. A significant disparity indicates overfitting.
- Analyze model outputs: Pay close attention to the LLM's outputs for signs of repetitiveness, nonsensical language, or biased responses.
- Regularly evaluate model performance on unseen data: Regularly test the LLM on unseen data to assess its generalization ability and identify potential overfitting issues.

Overfitting is a significant challenge for LLMs, but by employing various data augmentation, regularization, and training techniques, developers can build more robust and generalizable models. By carefully monitoring and evaluating LLM performance, we can identify and address overfitting early on, ensuring that these powerful language tools are used effectively and responsibly.

Remember, the specific solutions to overfitting will depend on the LLM's architecture, training data, and target applications. It's important to experiment and evaluate different approaches to find the most effective strategies for your specific LLM development needs.

1.2 LLM Application Anatomy

In the following section of this chapter, we delve into addressing the initial challenge of Precise Task Execution, exploring strategies such as prompt engineering, task decomposition, and fine-tuning.

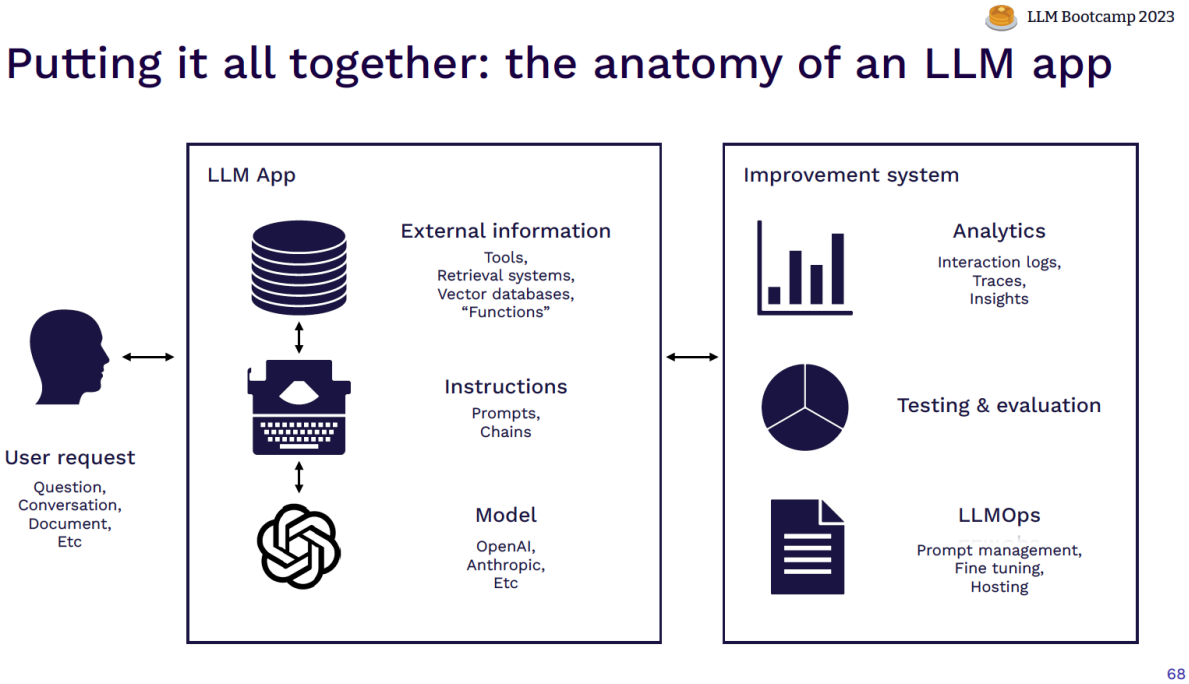
Subsequent chapters will tackle the remaining challenges in detail:

Chapter 2 specifically concentrates on enhancing language models through augmentation, various tools, and retrieval systems.

Following that, Chapter 3 examines the aspects of Scalability and Reliability in Production, emphasizing the roles of Large Language Model Operations (LLMOps) and Test Driven Development.

1.2 LLM Application Anatomy

[To-Do]: Add more context on LLM Application Architecture and Components



Retrieval Systems

Retrieval systems in the context of Large Language Model (LLM) applications refer to the mechanisms used to fetch and present information in response to user queries or requests. These systems are crucial for integrating LLMs like GPT-3 or GPT-4 into applications where accurate and relevant information retrieval is essential. Here's how they work in LLM apps:

- Query Processing: When a user inputs a query or request, the retrieval system interprets and processes this input, often using the LLM to understand the context and intent behind the query.
- Searching and Fetching Data: The system then searches through a large dataset or database to find relevant information. This dataset might be a structured database, a collection of documents, or the internet.
- Use of Embeddings and Vectors: Many advanced retrieval systems use vector representations of text (embeddings) generated by the LLM. These embeddings capture semantic meanings and relationships, allowing the system to find the most relevant information based on the query's context.

- Ranking and Filtering Results: Once potential answers or information are retrieved, the system ranks them based on relevance. This might involve additional processing by the LLM to evaluate the suitability of each piece of information.
- Presenting Information: The most relevant and useful results are then presented to the user. In some applications, the LLM may further refine or summarize this information for better clarity and usefulness.
- Feedback Loop: Many systems include a feedback mechanism where user responses to the retrieved information are used to improve future retrieval performance.

Vector Database

Vector database is a specialized database designed to efficiently store and retrieve high-dimensional vector data. These vectors are typically the embeddings generated by LLMs, which represent text, words, or even whole documents in a numerical form that captures their semantic meaning. The key aspects of a vector database in this context include:

- Efficient Storage of Embeddings: Vector databases are optimized to store the dense vector representations produced by language models. These embeddings are multi-dimensional and require specialized storage solutions for efficiency.
- Similarity Search: One of the primary features of a vector database is its ability to perform fast similarity searches. It can quickly find vectors that are closest (most similar) to a given query vector, which is essential for tasks like semantic search, recommendation systems, or even clustering.
- Scalability: Given the large size of datasets used in LLMs, vector databases are designed to scale effectively, handling large volumes of data without a significant loss in performance.
- Integration with AI Models: Vector databases are often integrated with AI and machine learning pipelines, allowing for seamless use of LLM-generated embeddings in various applications like chatbots, search engines, and more.

Function

In the context of an LLM (Large Language Model) application, a "Function" typically refers to a specific capability or task that the LLM can perform based on its training and programming. These functions are essentially different ways the LLM can be applied to solve problems or provide information. Examples of functions in an LLM app might include:

- Text Generation: Creating written content based on prompts or guidelines provided by the user.
- Question Answering: Responding to user queries with accurate and relevant information.
- Translation: Translating text from one language to another.
- Summarization: Providing concise summaries of longer texts.
- Sentiment Analysis: Determining the sentiment or emotional tone behind a piece of text.

- Chatbot Conversations: Engaging in dialogue with users, simulating a conversational partner.

Each of these functions leverages the LLM's understanding of language and context, demonstrating its ability to handle diverse language-based tasks. The design of an LLM app often involves tailoring these functions to meet specific user needs or business goals, providing a versatile tool for various applications.

LLMOps

LLMOPs, or Large Language Model Operations, refers to the practices and processes involved in deploying, maintaining, and managing large language models (LLMs) like GPT-3 or GPT-4. It encompasses a range of activities:

- Deployment: This involves setting up the infrastructure to host and run large language models, ensuring that they can handle high volumes of requests and operate efficiently.
- Scaling: Since LLMs often require significant computational resources, scaling them to meet user demand while optimizing for cost and performance is a key aspect of LLMOPs.
- Monitoring and Maintenance: Continuous monitoring for performance, accuracy, and reliability is crucial. Maintenance includes updating models with new data or improvements, and ensuring they operate within ethical guidelines and legal frameworks.
- Security and Privacy: Ensuring the security of the model and the privacy of the users' data is a critical part of LLMOPs, given the sensitive nature of the data LLMs can process.
- User Experience and Accessibility: This includes optimizing the interface and accessibility of LLMs for various applications, ensuring that they are user-friendly and effectively meet user needs.
- Compliance and Ethical Considerations: Ensuring that the deployment and use of LLMs are in compliance with relevant laws and ethical standards, especially considering the potential impact of their outputs.

LLMOPs is an emerging field, reflecting the growing role of large language models in various sectors and the need for specialized knowledge and practices to manage them effectively.

1.3 LLM System Design Templates

[Todo: Add a note that this is a personal opinion based template, based on my research summary and LLM bootcamps.]

2.1 LLM System Design Template (Short)

LLM System Design Template (Short)

1. Problem Formulation

- Define the Problem and LLM(s), clearly:
- User needs and challenges,
- LLM Limitations and challenges

2. Architectural Components

- Solutions Overview (to LLM limitations)
- LLM Model Architecture
- LLM Application Architecture

3. Data Needs

- Data Collection and Preparation
- Vector databases (if needed)

4. Feature Engineering

- Feature Selection

5. Model Selection and Development

- Model Selection
- Model Development (build, fine tune, etc)
- Prompt Engineering
- Model Testing (Test Data, Metrics)

6. IR System and Augmented LLMs

7. Deployment and LLMOps

- Deployment
- LLMOps (new concept, operations for LLMs)

8. Monitoring and Continual Improvement

- Monitoring
- Continual Improvement:
    - Feedback Loop:
    - Fine Tuning LLMs

9. Scaling

- Scale (Data volume, user requests)

10. Further considerations

- Integrations / Plugins
- Ethics and Compliance
- Explainability and Interpretability
- Further advancements: Explore new/future advancement (e.g. in multimodality, reason factual grounding).

2.2 LLM System Design Template (Long)

LLM System Design Template (Long)

1. Problem Formulation

- Define the Problem and LLM(s), clearly:
    - Target domain(s),
    - LLMs Purpose (persona, task),
    - Input(s) / output(s) (text generation, translation, dialogue, etc.) and their modal (text, code, audio, etc. - MultiModal LLM),
    - Data (privacy, access, and needs),
    - Scale (data and users),
- User needs (factual accuracy, creativity, engagement, etc.) and challenges,
- LLM Limitations and challenges in the chosen context

2. Architectural Components

- Proposed Solutions Overview (to LLM limitations)
- LLM Model Architecture:
    - Core architecture: (e.g. Transformer) - tied to Model Selection below
    - Model's input and output modalities
- LLM Application Architecture (🆕)
    - Prompt engineering, IR System and LLM Augmentation (if needed), D pipelines (Vector Database (if needed)), fine-tuning system (if need Tools, Chaining, LLMOps (Core and supplementary Ops), Deploym Platform, App sever and services

3. Data Needs

- Data Collection and Preparation
    - Data sources (web text, dialogues, code repositories),
    - Data cleaning, pre-processing, for quality and bias mitigation.
    - Data diversity and domain representation
- Vector databases (for embeddings management): Potential use for efficient storage retrieval of LLM embeddings.

4. Feature Engineering

- Feature Selection: Identify key features (tokenization, sentiment analysis, er recognition, etc), and their impact on LLM performance.

5. Model Selection and Development

- Model Selection
    - Trade offs between base models (proprietary vs. open-source)
    - Decision Factors: Task requirements and resources
    - Options: proprietary, open-source, in-house
- Model Development (build, fine tune, etc)
    - Model Training (if needed, mostly not):
        - HP tuning, optimizer, early stopping criteria, test data, etc
- Prompt Engineering (🆕)
    - Prompt engineering techniques (cloze deletion, few-shot learning, etc) for guid outputs and mitigating bias.
    - Prompt Iterations / templates / (training ?)
    - Prompt management (storage, versioning, etc)
- Model Testing (Test Driven Development)
    - Test Data
    - Metrics (Offline and Online, LLM-Specific and ML)
        - e.g. offline (BLEU, ROUGE, perplexity) and online (user satisfaction) .
    - (Benchmarking pipeline(s))

6. IR System and Augmented LLMs 🆕

- IR and LLM systems integration for factual grounding and context awareness
    - Augmented LLMs (hybrid models): leverage external knowledge for impro performance and domain adaptation.
    - benefits and challenges of IR system + LLMs integration

7. Deployment and LLMOps

- Deployment
    - Deployment platform (cloud vs on-premise)
        - Decision factors: latency, scalability, and cost constraints
    - Resource management
    - Serving infra
    - Service design
        - API Design (integrating LLM w/ other apps/systems)
        - Response/Output processing
    - Chaining and Tooling
        - Chaining multiple LLMs (complex tasks requiring multiple skill modalities).
    - Output format design, processing and validation
    - LLM Deployment challenges
- LLMOps (operations for LLMs, 🆕)
    - Core Ops (Prompt management, Fine-Tuning, Hosting Ops, Data, Eval & Test)
    - Supplementary Ops (Ethics and Compliance, API, Integration, User Support Ops

8. Monitoring and Continual Improvement

- Monitoring
    - Monitoring Signals (LLM performance metrics in real-world, user feedback)
- Continual Improvement:
    - Feedback Loop: iteratively improve the model based on data insights and u experience.
    - Fine Tuning LLMs

9. Scalability

- Scale for: Data volume and user requests

10. Further considerations

- Integrations / Plugins
- Ethics and Compliance
    - Responsible AI development practices
- Explainability and Interpretability
    - Improving interpretability: attention visualization, feature importance analysis, e
- Further advancements: Explore new/future advancement (e.g. in multimodality, reason factual grounding).

2. Prompt Engineering

[ToDo: Add more content]

Intro

Prompt: For us, a “prompt” is “text that goes into an LM”.
Prompt Engineering: “Prompt engineering” is the art of designing that text (not where that text comes from or where it goes).

Introduction to Prompt Engineering

- Essence of Prompt Engineering: A critical aspect of working with LLMs, involving the creation of input prompts that effectively guide the LLM to desired outputs.
- Impact on LLM Performance: Essential for determining how LLMs interpret and respond to queries, significantly influencing their effectiveness.

Techniques in Prompt Engineering

- Designing Effective Prompts: Focused on clarity, specificity, and contextual relevance. Clear and specific prompts guide the LLM more accurately.
- Utilizing Templates and Structures: Standardizing prompt templates for consistency; including structural elements like question phrases and bullet points to enhance response elicitation.
- Pipeline Development and Context Integration: Converting user requirements into understandable formats for LLMs, integrating relevant environmental information for timely suggestions.
- Snippeting and Prioritization: Selecting pertinent information for inclusion in the LLM's context; prioritizing crucial context elements for effective use.
- Model Selection: Balancing between speed and accuracy in model choice to ensure timely and useful outputs.

Advanced Prompt Engineering Strategies

- Conditional Prompts and Prompt Chaining: Tailoring prompts for specific outcomes; creating a sequence of prompts for complex tasks.
- Feedback Loops: Incorporating user feedback to enhance prompt relevance and accuracy.
- Chain of Thought Prompting: Guiding LLMs through a logical reasoning process for complex problem-solving.
- Zero-Shot and Few-Shot Learning: Enabling LLMs to perform tasks without explicit training on them.

Prompt Engineering in Practice

- Industry Case Studies: Demonstrating the application of prompt engineering across various sectors.
- Testing and Iteration: Methods for refining prompts based on user feedback and performance.

Challenges and Considerations

- Bias Mitigation: Strategies to avoid biases in prompts.
- Ethical Concerns: Ensuring prompts do not lead to harmful or inappropriate responses.

The Future of Prompt Engineering

- Automated and Interactive Prompts: Exploring AI-driven prompt creation and dynamic, user-responsive prompts.

Tools for Prompt Engineering

- Development Environments and Data Analytics Platforms: For real-time testing and feedback analysis.
- AI Prototyping Platforms: Like OpenAI's Playground or Google's Colab for experimenting with prompts.

Conclusion

- Evolving Field: Prompt engineering is dynamic, requiring continuous adaptation to keep pace with LLM advancements.

This comprehensive overview encapsulates the multifaceted discipline of prompt engineering, highlighting its crucial role in the functionality and ethical deployment of Large Language Models.

4.1. Prompting Intuitions

[TBD]

4.2 Prompting Techniques

4.2.1 [TBD]

[TBD]

4.2.X Prompting Templates

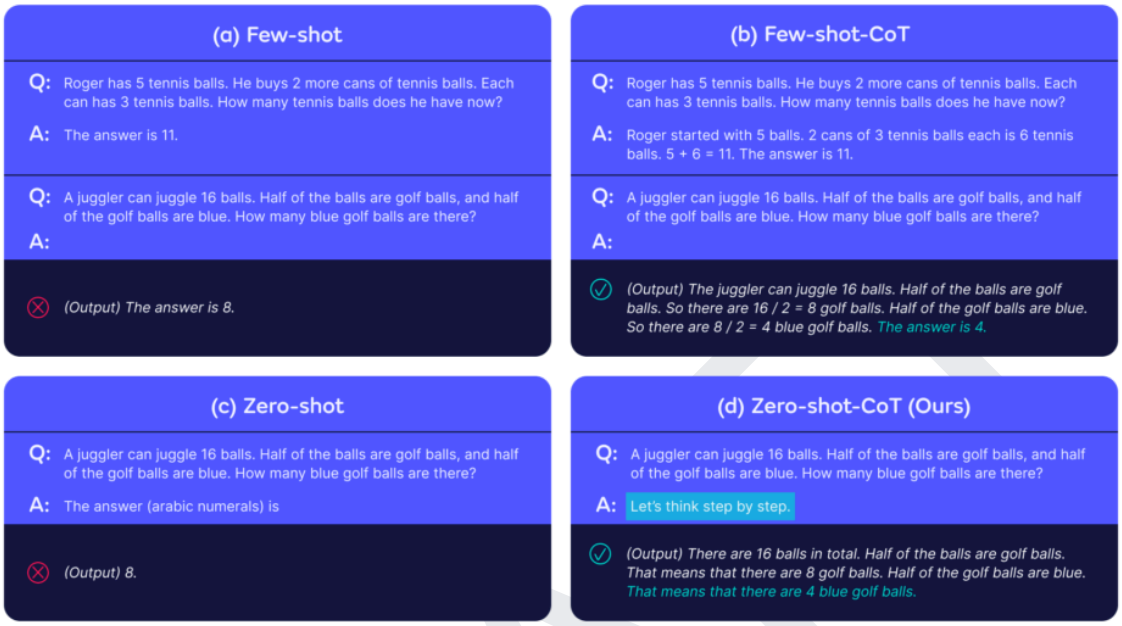


Fig. X.X Examples of prompting techniques

Much like the experimentation process that has evolved in traditional Machine Learning model training, prompt engineering has emerged as a core activity in LLM application development. There are several prompting techniques that lead to better results. The first technique is to write clear instructions and specify the steps required to complete the task. Most tasks can be done using a "**zero-shot**" prompt, but adding some examples with a "**few-shot**" prompt can further improve and tailor the response. Even simply including **instructive phrases** such as "show your work" or "let's think step by step" leads the model to iteratively develop a solution, increasing the chances of a correct answer. This "chain-of-thought" prompting is even more powerful if a few examples of reasoning logic are provided.

Prompts can then be templated and reused by an LLM application. They can be integrated into your code with a simple f-string or str.format(), but there are also libraries like LangChain, Guidance, and LMQL that offer more control. For chat completion APIs, there is generally a system prompt to assign the task, followed by alternating user and assistant messages. You can seed the chat with examples of user questions and assistant answers before running it. Experimenting and iterating on these prompt templates in a structured way will lead to improved model performance. The outputs of the model should be evaluated by either a scoring function or human feedback.

Prompting "Template" Creation Example

Let's dive into an example of creating a prompting template first:

```
   # Simple example of prompting an LLM with a template

   # Prompt template
   prompt_template = """
   This is a template for prompting an LLM. You can fill in the blanks with y
own information.

   {user_message}

   {assistant_message}
   """

   # User message
   user_message = "What is the capital of France?"

   # Assistant message
   assistant_message = "The capital of France is Paris."

   # Generate the full prompt
   full_prompt         =         prompt_template.format(user_message=user_messa
assistant_message=assistant_message)

   # Print the full prompt
   print(full_prompt)
```

Output:

```
   This is a template for prompting an LLM. You can fill in the blanks with your own
information.

   What is the capital of France?

   The capital of France is Paris.
```

This example demonstrates how a prompt template can be used to generate a consistent format for prompting an LLM. The template can be filled in with different information to create different prompts. This can be helpful for tasks such as chatbots, where you need to generate a lot of different prompts.

In addition to simple **f-strings** and **str.format()**, there are also libraries like **LangChain**, **Guidance**, and **LMQL** that offer more control over the prompting process. These libraries can be used to create more complex prompts that include things like conditionals and loops.

For chat completion APIs, there is generally a system prompt that assigns the task to the LLM, followed by alternating user and assistant messages. You can seed the chat with examples of user questions and assistant answers before running it. This can help the LLM to learn how to respond to different types of prompts.

Experimenting and iterating on your prompt templates is a great way to improve the performance of your LLM. By trying different things and seeing what works best, you can find the prompts that generate the most accurate and helpful responses.

The outputs of your LLM should always be evaluated by either a scoring function or human feedback. This will help you to identify any areas where your LLM needs improvement. By making changes to your prompts and training data, you can continue to improve the performance of your LLM over time.

Full Prompt Example for Chatbot Conversation System

Here's a complete example of defining a chatbot conversation system using the prompt template:

Application: A customer service chatbot for a clothing store.

Prompt Template:

```
   This is a conversation between a customer and a chatbot about cloth
products.

   Customer: {user_message}

   Chatbot: {assistant_message}
```

Seed Examples:

- **User** Message: "I'm looking for a new dress for a wedding." Assistant Message: "What style of dress are you interested in?"
- **User** Message: "I'm not sure. I'm open to suggestions." Assistant Message: "What colors do you like?"
- **User** Message: "I like red and blue." Assistant Message: "We have a great selection of red and blue dresses. What size are you?"
- **User** Message: "I'm a size 8." Assistant Message: "Okay, let me show you some dresses that might be a good fit for you."

- **User** Message: "Thank you!" Assistant Message: "You're welcome! Is there anything else I can help you with today?"

Full Prompt Example:

```
   This is a conversation between a customer and a chatbot about cloth
products.

   Customer: I'm looking for a new dress for a wedding.

   Chatbot: What style of dress are you interested in?

   Customer: I'm not sure yet. What styles do you have?

   Chatbot: We have a variety of styles, including ball gowns, A-line dresses,
cocktail dresses. We also have a selection of short and long dresses.

   Customer: I think I would prefer an A-line dress.

   Chatbot: Okay, that's a great choice. What color are you interested in?

   Customer: I'm open to suggestions. What colors do you think would
appropriate for a wedding?

   Chatbot: For a wedding, you could choose a classic color like black, white,
navy. You could also go for a more vibrant color like red, yellow, or green.

   Customer: I think I would like something a little more unique. Maybe someth
with a floral pattern?

   Chatbot: We have a beautiful selection of A-line dresses with floral patter
Would you like me to show you some of those?

   Customer: Yes, please!

   Chatbot: (displays a selection of A-line dresses with floral patterns)

   Customer: Wow, those are gorgeous! I think I found the perfect dress!

   Chatbot: I'm glad to hear it! Do you need any help with other accessories
shoes?

   Customer: No, I think I'm all set. Thank you so much for your help!

   Chatbot: You're welcome! I hope you have a wonderful time at the wedding!
```

Example User Interaction with Chatbot

Scenario: A customer wants to purchase a new dress for a wedding

Conversation:

- Customer: Hi, I'm looking for a new dress for a wedding.
- **Chatbot**: Hello! What style of dress are you interested in?
- **Customer**: I'm not sure yet. What styles do you have available?
- **Chatbot**: We have a wide variety of styles, including ball gowns, A-line dresses, cocktail dres and more. We also have a selection of short and long dresses.
- **Customer**: Do you have any A-line dresses?
- **Chatbot**: Yes, we do! We have a beautiful selection of A-line dresses in various colors and fabric
- **Customer**: I'm interested in something with a floral pattern. Could you show me some options?
- **Chatbot**: Absolutely! Here are some A-line dresses with floral patterns that you might like:



- Customer: Wow, these are all beautiful! I think I like the one with the blue and pink flowers.
- **Chatbot**: Excellent choice! What size are you interested in trying on?
- **Customer**: I'm a size 8.
- **Chatbot**: I'll have someone bring the dress in size 8 to your dressing room right away. Would like me to suggest any accessories or shoes to complement your dress?
- **Customer**: Yes, please! I'm open to suggestions.
- **Chatbot**: Based on your dress selection, I recommend these silver earrings and heels that w complement the floral pattern beautifully. Would you like to add them to your order?
- **Customer**: Yes, please! Thank you so much for your help!
- **Chatbot**: You're welcome! Is there anything else I can help you with today?
- **Customer**: No, that's all for now. Thank you again!
- **Chatbot**: You're welcome! I hope you have a wonderful time at the wedding!

Additional Notes:

- The **seed examples** provide the LLM with context and help it to understand the types of questions and answers that are expected in this conversation.
- The **full prompt** illustrates how the template can be used to generate a complete conversation.
- This example fills in the placeholders in the Prompt Template with actual conversation text. The code doesn't explicitly mention these **fields** ("user_message" and "assistant_message") because they are *implicitly understood by the format of the template.* Amazing, right?
- This example demonstrates how the prompt template can be used to define a chatbot conversation system. By providing the LLM with **context** and **examples**, you can create a chatbot that is able to have natural and engaging conversations with users.
- This is just a basic example, and the actual prompts used in a **real-world** chatbot would be much more **complex** and **specific**.

Chapter 3

Augmented Language Models

1. Intro to Augmented LMs

Challenge: **Limited Knowledge Scope**
There's a lot language models don't know

What (base) LLMs are good at:
- Language understanding
- Instruction following
- Basic reasoning
- Code understanding

What they need help with:
- Up-to-date knowledge (of the world)
- Knowledge of your data
- More challenging reasoning
- Interacting with the world

Context Window

Context window refers to the portion of the prompt that the model focuses on to generate a response. When you craft a prompt for an LLM, it includes various elements—background information, specific questions, instructions, etc. The context window within this prompt encompasses:

- Relevant Information: It's the part of the prompt that contains the most relevant information needed for the model to understand and respond accurately. This could be a specific question, a set of instructions, or key background details.
- Immediate Focus: The model uses the information within this window to form its response. Therefore, the content within the context window should be directly related to what you want the model to address or perform.
- Influence on Response Quality: A well-defined context window, where the relevant information is clear and concise, can significantly improve the quality and relevance of the model's responses. It guides the model on what to focus on.
- Managing Length and Clarity: Since LLMs have a limit to how much text they can consider at once (total context window), it's important to manage the length and clarity of the context window within your prompt. This ensures that essential information isn't overlooked or truncated due to length constraints.

Pros and cons:

- Context is the way to give LLM unique, up-to-date information.

- But it only fits a **limited amount of information**

LLM Bootcamp 2023

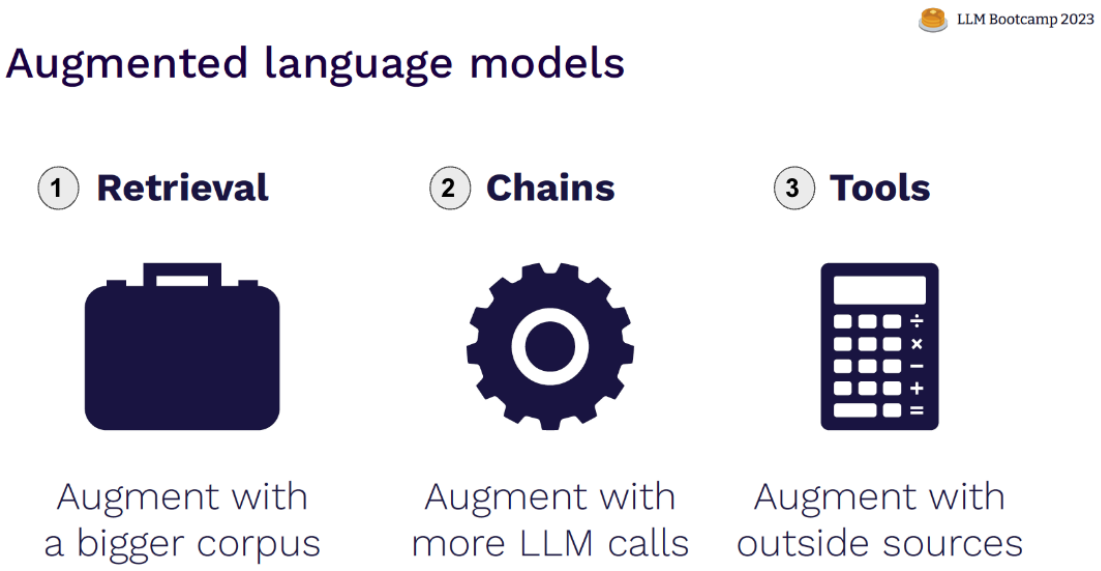
How much information can you fit in the context window?

Number of tokens	50	500	4,000	32,000	256,000	2,048,000	8,192,000	65,536,000
Example model		GPT-1	GPT-3.5	GPT-4-32K				
How much is it?	A sentence	~4 paragraphs of writing	New Yorker article	College thesis	Novel	~7,500 emails (or, about 1 year's worth for a productive office worker)	30 seconds worth of tweets (At 40 tokens per tweet, ~400000 / minute)	~500Mb of unicode text data (a single ElasticSearch node can store 50gb)

- Context windows are growing fast, but won't fit everything for a while
  (Plus, more context = more $$$)

Augmented Language Models

How to make the most of a limited context by augmenting the language model?



2. Retrieval Augmentation

2.1 Why Retrieval Augmentation?

Say we want our model to have access to user data, Approach 1: put it in the context!
What if we have thousands of users?

- Context-building is information retrieval.
- **Information retrieval**: the process of obtaining **relevant information** from a **large collection of documents** in response to a specific **query** or need. It involves identifying, indexing, and retrieving documents or other data resources that match the criteria specified in a query.
    - Searches: full-text, semantic

2.2 Traditional Information Retrieval

Information Retrieval Basics

- Query. Formal statement of your information need. E.g., a search string.
- Object. Entity inside your content collection. E.g., a document.

• Relevance. Measure of how well an object satisfies the information need
• Ranking. Ordering of relevant results based on desirability

Traditional information retrieval

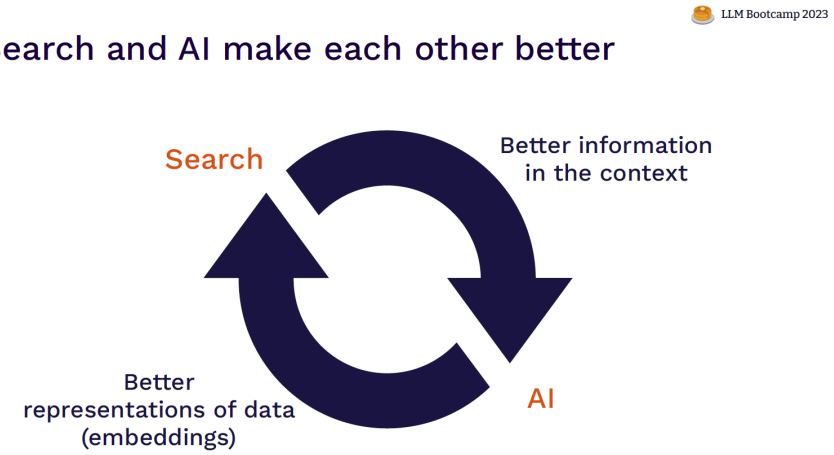
- **Search** via Inverted Indexes
- **Relevance** via boolean search
    - E.g., only return the docs that contain: simple AND rest AND apis AND distributed
    - AND nature
- **Ranking** via BM25. Affected by 3 factors
    - Term frequency (TF) — More appearances of search term = more relevant object
    - Inverse document frequency (IDF) — More objects containing search term = less important search term
    - Field length — If a document contains a search term in a field that is very short (i.e. has few words), it is more likely relevant than a document that contains a search term in a field that is very long (i.e. has many words).

Search engines

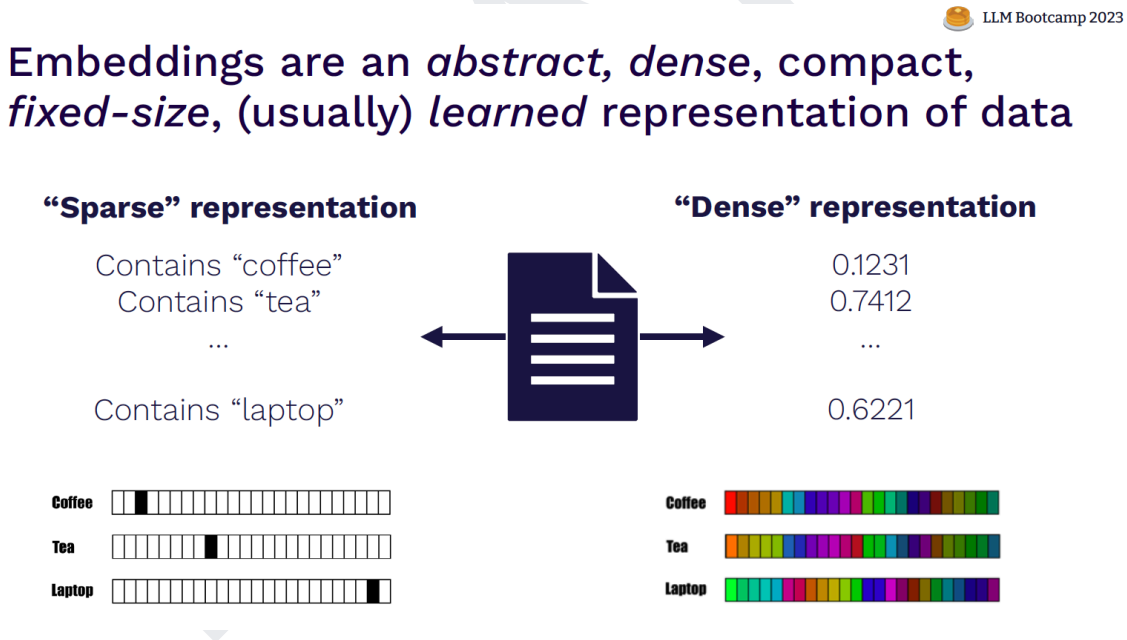
More than inverted indexes:
- Document ingestion
• Document processing (e.g., remove stop words, lower case, etc)
• Transaction handling (adding / deleting documents, merging index files)
• Scaling via shards
• Ranking & relevance
• Etc

Limitations of “sparse” traditional search
- Only models simple word frequencies
• Doesn’t capture semantic information, correlation information, etc

2.3 Information Retrieval via Embeddings (AI-powered)



2.3.1 Embeddings



What embeddings are NOT?

- Embeddings are not restricted to one modality of data
- Embeddings are not restricted to early layers of a neural network
- Embeddings are not necessarily one of the last learned layers of a network
- Embedding layers are not fundamentally different types of neural network layers

• Embeddings don't have to use embedding layers as defined in frameworks like PyTorch or Keras
• Embeddings don't have to specifically refer to an atomic unit
• A single embedding doesn't have to refer to one single type of input
• Embeddings don't have to be from a neural network at all
• Embeddings don't have to be directly comparable in vector space

Why embeddings?
Vectors are a compact, universal representation of data

What makes a good embedding?

- Ultimately, utility for the downstream task
    - Use your task!
    - If you can't, pick a broad benchmark (e.g. Massive Text Embedding Benchmark (MTEB) by Huggingface)

- Some other desiderata:
    - Close things should be close, far things far
    - Vector math

Embedding Choices

- A simple embedding: Word2Vec
    - Try to predict the center word from the surrounding context (CBOW model), or vise versa (Skip-Gram).
- A solid baseline: Sentence Transformers
    - Cheap / fast to run
    - • Widely available
    - • Works decently well
    - Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks
        - Siamese BERT: an adaptation of the BERT model used in a Siamese network architecture for tasks involving the comparison of text pairs, such as sentence similarity and paraphrase identification.
        - Twin Networks: The Siamese architecture uses two identical subnetworks with shared weights, and each subnetwork is a BERT model.
- Good, fast, and cheap: OpenAI embeddings
    - Use **text-embedding-ada-002**
        - Sentence embedding model
        - Developed as part of the GPT-3 family.
        - Near-SoTA
        - Easy to use, good results in practice
- SOA embedding: Instructor
    - One Embedder, Any Task: Instruction-Finetuned Text Embeddings

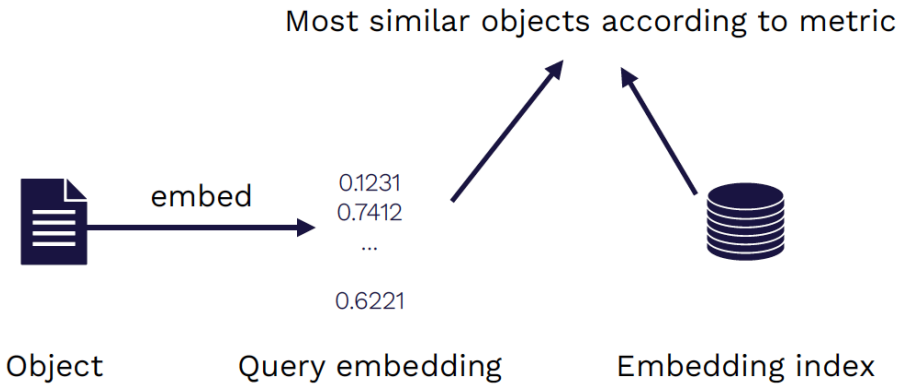
- Prepend the task description to the text, then embed it
- • At test time, describe a new task, prepend the description, get new
- embeddings

Note on sentence embedding:
Adopting a GPT model, like GPT-3, for sentence embeddings involves repurposing its generative capabilities to produce fixed-length vector representations that capture the semantic content of sentences. This involves training on diverse texts and mean pooling token embeddings to create fixed-length vectors that represent entire sentences. These models, initially designed for text generation, are fine-tuned to understand sentence-level semantics, making them suitable for applications like semantic search and text classification.

2.3.2 Embedding Relevance and Indexes

Relevance with embeddings

Finding the most similar objects according to a similarity metric



Similarity metrics

- Cosine similarity
- Dot product
- Euclidean distance
- Hamming distance

OpenAI recommends using cosine similarity. The choice of distance function doesn't matter much, as the embeddings are normalized to length 1.

Nearest neighbors search

A **minimal recipe** for nearest neighbor search:

- Embed your corpus
- Store embeddings as an array
- Embed the query, compute dot product with the array

```
# vec -> 1D numpy array of shape D
# mat -> 2D numpy array of shape N x D
# k -> number of most similar entities to find.
similarities = vec @ mat.T
partitioned_indices = np.argpartition(-similarities, kth=k)[:k]
top_k_indices = partitioned_indices[np.argsort(-similarities[partitioned_indices])]
```

When do you need more?

- If you have <100K vectors or so, you probably won't notice the difference in speed
- Above a certain scale, it does matter!

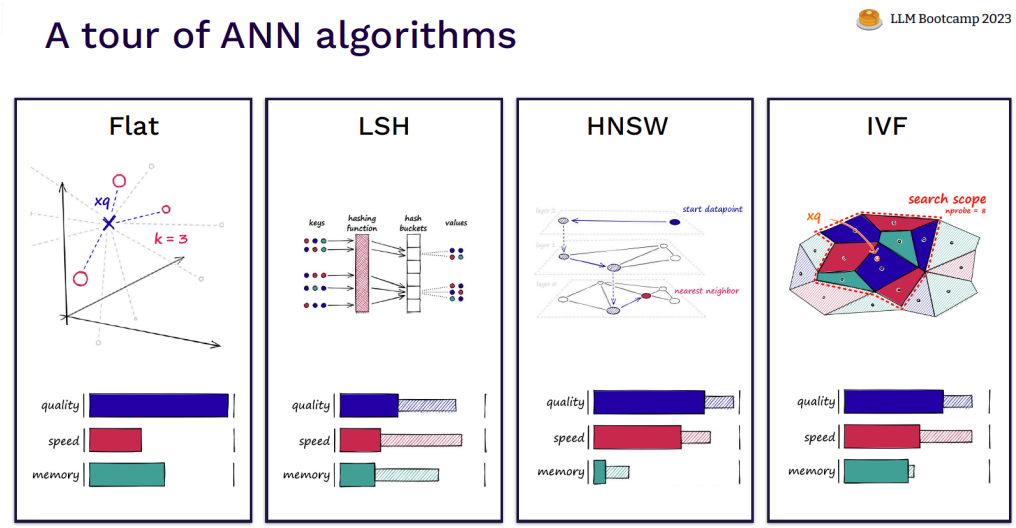
How do we scale similarity search?

via *approximate nearest neighbor indexes (ANN)* aka **embedding indexes**.

Embedding indexes

- Data structures that let us perform approximate nearest neighbor search
- Different index types are available: tradeoffs between speed, scalability, and accuracy

ANN Algorithms



[Algos Details TBD]

Embedding index tools

- Facebook AI Similarity Search (FAISS)
- CPU + GPU
- Supports lots of algos
- Hnswlib
- Easy to use HNSW implementation
- nmbslib
- More performant HNSW implementation
- Annoy
- Very simple + easy but lower performance

Which index should I choose?

- When prototyping, it doesn't matter. Use numpy if you want
- When productionizing, the much more important choice is not the index, but the IR system it's part of
- If you must choose, FAISS + HNSW is a reasonable start

Limitations of ANN indexes

They're just a data structure. They do not offer:

- Hosting
- Storing data / metadata alongside vectors
- Combining sparse + dense retrieval

- Managing the embedding functions themselves
- Vertical / horizontal scaling
• Beyond prototyping, you'll want an IR system / database that supports more of these

2.3.3 Embedding Databases (aka Vector Databases)

Challenges with AI-powered Information Retrieval

An ideal IR System

Dump in a bunch of data	Run a query	Get the most relevant data back
	*Descriptions of recent concerts in the bay area*	[{"document_id": "12345", "snippet": "..."}, ...]

Challenges

- Dumping in a bunch of data:
    - Database: scale, reliability
    - Embedding mgmt (e.g. what embedding to use, how to change, etc)
    - Document splitting
        - When the document is too long to fit in the context for the embedding model
            - Pick a separator ("\n")
            - Split text by separator up to max size chunks
            - Advanced: try to make the chunks more semantically consistent
- Run a query
    - Query language
        - There's no query document; there's a query language instead
- Get the most relevant data back
    - Search algorithm
        - Dealing with hierarchical structure in the index
            - If all knn's are chunks from the same doc,
            - If all docs are from same corpus,
            - If docs are old, etc

Embedding Databases ()

LLM Bootcamp 2023

Managed embedding databases

Tool	Company funding	Prominent users	DB features	Embedding mgmt	Advanced querying	It's for...
Chroma	$17M	N/A	✓	✓	✓	Betting on the most "AI-native" tool in the category
Milvus	$60M	ebay, Walmart	✓	✗	✓	Scale & enterprise
Pinecone	$38M	shopify, GONG	✓	✗	✗	Fastest to get started
vespa	N/A	yahoo!, Spotify R&D	✓	✓	✓	Battle-tested; most powerful
weaviate	$18M	N/A	✓	✓	✓	Embedding mgmt and flexible GraphQL-like query interface

Do we need an embedding database?

- Elasticsearch, postgres, redis can be configured to run exact / approximate nearest neighbor search
- You get all of the benefits of those platforms, with reasonably performant KNNs
- Won't work for the most complicated queries or highest scale, but will probably work for you

Recommendations for Embedding Databases

- When you are ready to move on from prototyping, move to a DB you are using already (Postgres / elastic / redis)
- If you don't have one of those databases, then move to **Pinecone** for speed of setup

Embedding database API call example

Below is an example showcasing the typical API structure of a vector database, like Pinecone or a similar service. This example will demonstrate basic operations such as adding vectors, querying for similar vectors, and updating or deleting vectors.

Note: This is a hypothetical example for illustrative purposes.

```
# Import necessary libraries
from language_model_client import LanguageModel
from vector_database_client import VectorDatabase

# Initialize the language model and vector database clients
lm = LanguageModel(api_key='language-model-api-key')
db = VectorDatabase(api_key='vector-database-api-key')

# Create a new index in the vector database
db.create_index(name='text_embeddings_index', dimensions=768)  # assuming 768-dimensional embeddings

# Example texts to generate embeddings for
texts = ["Hello world", "Machine learning is fascinating", "Exploring AI capabilities"]

# Generate embeddings for each text using the language model
for text in texts:
    embedding = lm.get_embedding(text)
    document = {"id": text, "vector": embedding}
    db.insert(document, index_name='text_embeddings_index')

# Querying the index for similar vectors
query_text = "AI technologies"
query_embedding = lm.get_embedding(query_text)
query_results = db.query(
    query_vector=query_embedding,
    index_name='text_embeddings_index',
    top_k=2  # return top 2 similar vectors
)

# Displaying the query results
for result in query_results:
    print(f"Text: {result['id']}, Similarity Score: {result['score']}")

# Cleanup: delete the index
db.delete_index(name='text_embeddings_index')
```

In this example:

- We initialize both the language model and vector database clients.
- A new index is created in the vector database for storing text embeddings.
- We generate embeddings for each piece of text by making an API call to the language model.
- These embeddings are then stored in the vector database with their associated text.
- For querying, we generate an embedding for a query text and use it to find similar texts in the database.
- The results show which texts are most semantically similar to the query.

Here's an example of what the output might look like for the line print(f"Text: {result['id']}, Similarity Score: {result['score']}"):

```
Text: Machine learning is fascinating, Similarity Score: 0.89
Text: Exploring AI capabilities, Similarity Score: 0.73
```

In this example output:

- The result['id'] refers to the id of the document in the database, which in our case is the original text itself.
- The result['score'] is a numerical value representing the similarity score between the query text and the document in the database, typically ranging from 0 (no similarity) to 1 (identical).

This is a more practical approach, demonstrating how embeddings can be generated and utilized in real-world applications. The actual implementation would depend on the specific APIs and libraries used.

Chapter 4

Model Selection and Development

1. Model Selection

1.1. Base Model Selection

Model Selection Trade-offs

The best model for your use case depends on tradeoffs between:

- Out-of-the-box quality for your task
- Inference speed / latency
- Cost
- Fine-tuneability / extensibility
- Data security and license permissibility

Most use cases, most of the time: start with GPT-4

Proprietary or open-source?

Proprietary models are better...

- Higher quality today
- Serving open source models introduces infrastructure overhead
- Cheaper out-of-the-box

... Unless you really need OSS

- Much easier to customize
- Respect data security
- Cheaper in the limit

Measuring performance of LLMs

- The only way to know which LLM will work best is to evaluate it on your task
- Benchmarks can be helpful, but are also misleading

Recommendations for Base Model

- Most projects should start with GPT-4

- This will give you a proof of concept about the feasibility of your task (Metaphor: “prototype in Python”)
- If cost or latency is a factor, consider “downsizing”
    - GPT-3.5 and Claude are good choices and comparable in performance
    - If you want to go even faster / cheaper, any provider will do, but Anthropic’s option is the most “modern”
- OSS is a viable option, but is a lot more work
- Mistral-7B and LLaMA 2 are viable options

2. Development (Test Driven)

1.2 Iteration and prompt management

Question: As you work on your prompts and chains, how to ‘save your work’?

Prompt engineering today

- Every time I change my prompt, I play around with it in a playground
- Old prompts are lost to time
- No way to reproduce experiments, share work with the team, etc

Three levels of prompt/chain tracking

- Level 1: do nothing (e.g., just make prompts in the OpenAI playground)
    - Good enough for v0, not what you want for building apps
- Level 2: track prompts in git
    - What you should do most of the time
- Level 3: track prompts in a specialized tool
    - For running parallel evals, decoupling prompt changes from deploys, or involving non-technical stakeholders

What to look for in a specialized prompt tracking tool?

- Decoupled from git
- Logged prompts are executable both in code and UI
- Connected to execution visualizations
- Deploy directly from the tool

Prompt management recommendations

- Manage your prompts and chains in git

- If you either collaborate with non-technical stakeholders, or automate your eval, then it's worth trying one of the experiment mgt tools (or keeping an eye out for new ones

1.3 Testing

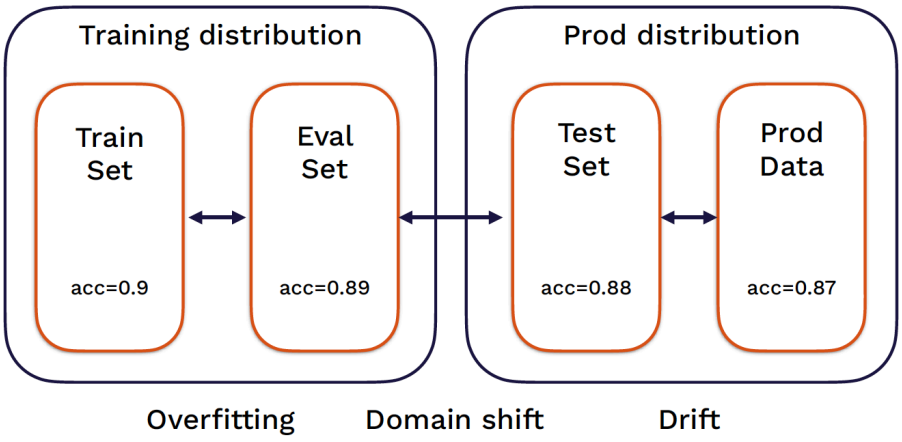
- LLMs make tons of mistakes

• Just because your new prompt looks better on a few examples
does not mean that it's better in general
- Super common to improve in one way and get worse in another!

How to **measure** whether your **new model / prompt** is better than the old one?
This boils down into two main questions:

1. What **Data**?
    - Determine the type of dataset needed for evaluation, considering factors like diversity, relevance to the task, and alignment with the expected real-world data the model will encounter.
2. What **Metric(s)**?
    - Select appropriate metrics based on the specific objectives of your model; for instance, accuracy and precision for classification, or perplexity and ROUGE scores for LMs in NLP applications etc.

Traditional ML Model Testing



A refresher on the above concepts

- **Overfitting:** Overfitting occurs when a model learns the training data *too well*, including its *noise and outliers*, leading to *poor generalization* to new data. This is typically manifested as a mismatch in performance between the *training set* and the *evaluation set*.

- **Domain Shift:** Domain shift refers to the situation where a model *trained* on one *data distribution* performs poorly on a different data distribution. This is often observed when moving from the *training distribution* to the *production distribution*, where real-world data may differ from the training data.

- **Drift:** Drift, or more specifically *concept drift* and *data drift*, happens when the *data* or the *relationship between input and output* variables *changes over time*. This leads to a gradual degradation in model performance, most notably in the *production set*, as it diverges from the data the model was originally trained on.
    - Example (online shopping behavior): data drift occurs as the e-commerce platform's user demographics and purchasing trends evolve over time, leading to a decrease in the accuracy of an ML model originally trained on older data that no longer reflects current user behavior and preferences.

A Side Note (Test Set Selection):

In ML, the test set is ideally selected to reflect the production distribution, not the training distribution. This ensures the test set accurately represents the real-world data the model will encounter in production, providing a better assessment of the model's generalization to new, unseen data.

In practice, however, this is not not always achievable due to various constraints:

- Data Availability: Sometimes, the exact type of production data might not be available at the time of model development, especially in rapidly changing or evolving domains.
- Data Evolution: Production data can evolve over time due to trends, user behavior changes, or external factors, leading to a mismatch with the initially captured test set.
- Resource Constraints: Collecting and maintaining a test set that continuously aligns with the production distribution can be resource-intensive.
- Privacy and Access Issues: In some cases, especially with sensitive or proprietary data, there might be restrictions on using real production data for testing purposes.

Therefore, while it's a best practice to make the test set as representative of production data as possible, various factors can lead to deviations from this ideal in real-world ML applications.

Why doesn't this work for LLMs?

- You don't have access to the training distribution

- Production distribution is *always* different than Training (Trained on the internet ➡ there is always drift, it doesn't matter as much)
- Qualitative: It's hard to define quantitative metrics
- Diversity of behaviors: It's hard to summarize a diverse set of inputs and tasks with a single number (metric).

1.3.1 Test Data

Goal: Building an evaluation dataset for *your* task.

1. **Start incrementally**
    - Start by evaluating ad-hoc
    - As you find "interesting" evaluation examples, organize them into a small dataset
        - To evaluate, run your model on every example in the dataset
        - What makes an "interesting" example?
            - Hard, Different
2. **Use your LLM to help**

    LLMs can help you generate test cases!

3. **Add more data as you roll out**

Hard data

"Hard data" here refers to challenging or difficult data examples that a model needs to learn from or improve upon. For example:

- What do your users dislike? - This involves collecting data on user preferences or complaints, which can be challenging because it often includes varied, unstructured, and sometimes ambiguous feedback.
- What do your annotators dislike? - This pertains to the data points that annotators find difficult or problematic, which could be due to complexity, ambiguity, or other factors that make them hard to label or categorize accurately.
- What does another model dislike? - This involves identifying data that other models struggle with or misinterpret, which is useful for understanding limitations or blind spots that might exist in current modeling approaches.

Different data

The following examples of "different data " indicate a focus on diversity and coverage in the data set:

- Outliers relative to your current eval set: This refers to data points that are significantly different from the majority of the data in the current evaluation set. Including outliers helps

ensure that the model can handle rare or unusual cases, not just the typical or average scenarios.

- Underrepresented topics, intents, documents, etc: This involves incorporating data related to topics, intentions, or document types that are not adequately represented in the current data set. By doing so, the model's ability to understand and respond to a wider range of subjects and user intents is improved, enhancing its robustness and reliability.

4. Toward "Test Coverage" for LLMs

Test Coverage in AI

In the context of AI, particularly in ML model development, "test coverage" refers to the extent to which a model's testing process evaluates the model across various scenarios and data points. This includes:

- **Variety of Data**: Ensuring the test dataset covers a wide range of inputs, from common scenarios to edge cases, to validate model performance across different situations.
- **Diverse Scenarios:** Testing the model in a variety of real-world and hypothetical scenarios to assess its reliability and robustness.
- **Gap Analysis and Benchmarking:** Using diversity metrics, gap analysis of model failures, and benchmarking against industry standards to ensure comprehensive coverage and address potential data diversity issues.

1.3.2 Evaluation Metrics

When evaluating LLMs in production, a blend of offline and online metrics is employed:

Offline Metrics

1. **Regular ML Metrics:**
    - **Accuracy**: Measures correctness in predictions.
    - **Perplexity**: Assesses avg uncertainty of the model in predicting the next token (prediction capability), the lower the better
    - **F1 Score**: Balances precision and recall for classification tasks.
    - **GPU Utilization**: Tracks tokens used for cost estimation.
    - **Responsible AI**: Monitors content filtering and ethical compliance.
        i. % of Prompts with HTTP 400 Error.
    - **Ethical AI:** measure how well the models adhere to ethical standards, such as **fairness**, **inclusivity**, and **transparency**.
        i. Examples: Fairness Audit Score, Transparency Index, and Inclusion Ratio
2. **Reference Matching Metrics:**
    - **BLEU Score**: For machine translation similarity.

    - **Semantic similarity** and **consistency** checks using another LLM.
3. **Static Metrics:**
    - Verification of output **format** and structure, like JSON.
    - Model-based **scoring** of answers (1-5 scale).

Online Metrics:

1. **Performance Metrics (Speed):** Time to first token render (from the time user submits the prompt), request rate (handled prompts/sec), and token rendering speed (tokens/sec).
2. **User Experience Metrics:** User engagement, satisfaction, and retention.
3. **Productivity Metrics**: Assesses AI-generated content's impact on collective productivity, e.g. Time Saved (per task), or No. of tasks executed within a timeframe, etc
4. **A/B Testing**: Evaluates impact of LLM features on **user value** (e.g. user interaction, satisfaction, etc) and **cost efficiency** (cost implications of implementing and using a new model/feature).
5. **Launch Experiments:** run at the time of product release to ensure feature performance and reliability; They Include:
    - **Dark mode experiments**: deploying a new feature or service on the prod server w/o exposing it to users. e.g. a new recommendation algorithm is running on product server and making recommendations using real user data but these recommendations are not shown to the user and are just logged for analysis.
    - **0-1 experiments:** controlled rollouts where a feature is introduced to a small, selected group of users (experimental group, 1), vs control group, (0) to measure its impact.

These metrics collectively ensure that LLMs are robust, user-friendly, and ethically aligned with responsible AI practices.

LLM Metric Selection

The choice of metrics for LLM evaluation is guided by the availability of correct answers (e.g. use accuracy or F1), reference answers (e.g. use reference matching metrics like BLEU score), and previous answers (e.g. consistency check), as well as the presence of human feedback (e.g. use user experience metrics).

A pivotal strategy involves **using LLMs to evaluate the performance of other LLMs**.

Automatic Evaluation

Question: Can we evaluate LLMs automatically?
Answer: As of April 2023, the automatic evaluation of LLMs is possible but remains a **complex and nuanced challenge**. There are several methods used for automated evaluation, each with its own strengths and limitations:

- **Automated Metrics:** Metrics like BLEU, ROUGE, and perplexity are commonly used for tasks like translation and text generation. However, these metrics often fail to capture the full scope of language understanding and generation capabilities, especially in more open-ended or creative tasks.
- E**mbedding-Based Evaluation:** Methods that compare embeddings of the model-generated text with reference text can provide insights into semantic similarity. But these too can miss nuances like style, tone, or context appropriateness.
- **Task-Specific Benchmarks:** Benchmarks designed for specific tasks (like GLUE for understanding, or SuperGLUE) provide automated evaluation frameworks. They are useful but often don't cover the entire spectrum of capabilities that LLMs offer.
- **Model-Based Evaluation:** Using other AI models to evaluate the outputs of LLMs is an emerging area. For instance, using one model to generate text and another to assess its quality.
- **Continuous Monitoring:** For models in production, automated monitoring tools can track performance indicators like user satisfaction, engagement metrics, or error rates.

While these methods provide valuable insights, they often need to be supplemented with human evaluation to fully assess aspects like coherence, creativity, factual accuracy, and adherence to social norms and ethical considerations. The field continues to evolve, with ongoing research aimed at developing more comprehensive and accurate automated evaluation methods for LLMs.

Update in November 2023

As of November 2023, the automatic evaluation of LLMs continues to be a complex and multifaceted task.

While some advancements have been made, the fundamental challenges and limitations remain.

Current evaluation methods still rely on a mix of **automated metrics** like **perplexity** and **human evaluations**, each with its own limitations.

- Metrics like perplexity don't fully capture semantic quality, coherence, or factuality, and human evaluations, while indispensable, are costly and difficult to scale.
- Additionally, automatic diversity metrics are still a focus of active research, aiming to better capture attributes like creativity and novelty in LLM outputs.

Overall, while existing methods provide valuable insights, they have limitations such as training data leakage, narrow metrics, costly human assessment, limited reference data, lack of diversity measurement, narrow benchmarks, and susceptibility to adversarial inputs

Task Specific Metrics

LLM Metrics for Different Tasks:

Task	Common Metrics	Description
Text Generation	BLEU, ROUGE-L, ROUGE-N	Measure overlap between generated text and reference te (higher is better).
Text Summarization	ROUGE-1, ROUGE-2, ROUGE-L, Pyramid ROUGE	Assess coverage and fluency of summaries compared to source documents.
Machine Translation	BLEU, Chrf, TER, WER	Evaluate translation quality by comparing generated translations to references (lower is better for TER and WE
Dialogue Systems	BLEU, ROUGE, Perplexity, Human Evaluation	Assess fluency, coherence, relevance, and informativeness conversation responses.
Question Answering	Accuracy, F1-score, MRR	Measure ability to answer factual questions correctly and retrieve relevant information.
Text Classification	Accuracy, Precision, Recall, F1-score	Evaluate ability to correctly categorize text samples into predefined categories.

Additional notes:

These are just some common examples, and specific metrics may vary depending on the specific task and desired qualities.

Some tasks may require custom metrics tailored to their unique challenges.

Human evaluation is often also used to assess LLM performance, especially for tasks where automatic metrics are not reliable.

LLM Public Benchmarks

- 🤗 Open LLM Leaderboard
- *Eleuther AI) Language Model Evaluation Harness

Chapter 5

Deployment, LLMOps, and Scaling

1. Deployment

Deploying LLMs: Challenges

Despite their immense potential, deploying LLMs in production environments presents unique challenges:

- **Model Selection and Customization:** Choosing the right LLM for a specific task requires careful consideration of performance, cost, and customization options. Building vs. buying vs. open-source models each offer advantages and drawbacks.
- **Prompt Engineering:** Crafting effective prompts that guide the LLM towards the desired output is crucial for accurate and efficient performance.
- **Resource Management and Optimization:** LLMs can be computationally expensive to train and run, requiring efficient resource allocation and optimization techniques like quantization and pruning.
- **Safety and Security Concerns:** LLMs are susceptible to biases, misuse, and manipulation, necessitating robust safeguards and ethical considerations.

Solutions for LLM Deployment:

To address these challenges and unlock the full potential of LLMs, innovative solutions are emerging:

- **Orchestration Platforms:** Tools like Flyte and Vertex AI streamline the deployment and management of LLMs, simplifying workflows and ensuring efficient resource utilization.
- **Cloud Platforms:** Services like Azure and Google Cloud offer managed solutions for deploying and running LLMs, providing scalability and flexibility.
- **LLMOps:** This emerging discipline provides a comprehensive framework for LLM operations, covering the entire lifecycle from development to monitoring and governance.
- **Continuous Monitoring and Improvement:** Continuously monitoring LLM performance, incorporating user feedback, and regularly retraining models are crucial for maintaining accuracy, fairness, and effectiveness.

2.1 Deploying LLMs

- Just call the API from your frontend
• Where it becomes more complicated
- If you have significant logic beyond the API call (complicated

prompt construction, complicated chains).
• Might want to isolate as a service
• Deploying open-source LLMs is a whole other thing

Deploying Open Source LLMs
• Beyond our scope
• Lots of frameworks emerging
- Ray / Anyscale
- Mosaic
- Huggingface
- Startups like Base10 etc
• Some references below

More info in [How to train your own Large Language Models](#).

- Self-critique

- Ask an LLM “is this the right answer”
• Sample many times, choose the best option
• Sample many times, ensemble

Output Validation System

An LLM output validation system is a tool designed to assess and ensure the quality, accuracy, and appropriateness of the responses generated by LLMs. It functions by applying a set of predefined criteria or rules, such as checking for factual accuracy, adherence to ethical guidelines, or alignment with specific application requirements. This system acts as a safeguard, enhancing the reliability and trustworthiness of LLM outputs, especially in applications where accuracy and context sensitivity are crucial.

Example: Guardrails AI

Guardrails AI, is an open-source library for creating custom validators and orchestrating the prompting, verification, and re-prompting process.

It includes a library of commonly used **validators** for multiple use cases and a language for effectively communicating requirements to LLMs. This system acts as a sort of 'firewall' around LLM applications, defining and enforcing assurance parameters, and is especially useful in scenarios where LLM outputs need to be regulated according to specific guidelines or quality metrics.

As an example, consider an LLM being used for customer service chatbots, and customer asks about a refund policy. Validators, such as in Guardrails AI will check the chatbot's response for compliance with *actual policy*, *accuracy in info*, and *appropriateness in tone*.

Improving LLM output quality in production

Self-critique
- Ask an LLM "is this the right answer"
• Sample many times, choose the best option
• Sample many times, ensemble

2.2 Chains

• Sometimes the best context for your LLM doesn't exist directly in your corpus
• Instead, the best context for your LLM might be the output of another LLM!

Example patterns for building chains

The QA pattern
- Question ➡ embedding ➡ similar docs ➡ QA prompt
• Hypothetical document embeddings (HyDE)
- Question ➡ document generating prompt ➡ rest of QA chain
• Summarization
- Document corpus ➡ apply a summarization prompt to each ➡ pass all document summaries to another prompt ➡ get global summary back

Tools for building LLM chains

LangChain
One of the fastest growing OSS projects of all time
• Python + JS
• Alternatively, many people just roll their own

2.3 Tools

In the context of enhancing LLMs with additional capabilities, **Tools** refer to *external functionalities or systems* that *LLMs can interact with to extend their abilities* beyond basic text generation and understanding. Examples like Tool Augmented LMs, Toolformer, WebGPT, and ChatGPT Plugins represent this concept. Here's a brief overview:

Tool Augmented LMs:

These are LLMs augmented with the ability to use external tools or services. For example, an LLM might call a web API to fetch real-time data or use a calculator for complex arithmetic.

Synergizing Reasoning and Acting in LMs

This approach involves integrating reasoning (understanding and planning) and acting (executing tasks using external tools) within LLMs, enhancing their problem-solving capabilities.

Example: AI Assistant
A practical example of this can be illustrated in an advanced AI assistant application. Consider a scenario where a user interacts with an LLM-based AI assistant for travel planning:

User: "I need to plan a trip to Paris for next week. Can you help?"

AI Assistant: LLM with Reasoning and Acting Capabilities

Reasoning:
The LLM first processes and understands the request. It reasons about the necessary steps: checking flight availability, dates, prices, hotel bookings, and tourist attraction information.

Acting:
The LLM then engages with external tools and databases. It might use an API to access a flight booking service to find available flights to Paris for the specified dates and another service to check for hotel availability.
It could also interact with a tourist information database to compile a list of recommended attractions or events happening in Paris during that week.

Toolformer

Toolformer refers to a concept where LLMs are trained to understand when and how to use external tools to complete tasks. The LLM learns to interact with these tools as part of its response generation process, effectively 'teaching itself' to leverage external functionalities.

WebGPT

WebGPT extends the capabilities of LLMs by enabling them to interact with web content. It can browse the internet, retrieve information from web pages, and use this information to inform its responses.

ChatGPT Plugins:

ChatGPT can be enhanced with plugins, which are like add-ons that provide additional capabilities. These plugins allow ChatGPT to interface with external databases, software, or services, enabling it to perform a wider range of tasks and provide more detailed and specific responses based on external data sources and tools.

- Provide API spec and description so the model knows how to use it
- Description is passed in a system message to ChatGPT
- Model can choose to invoke the API and include the results in the response

```
{
    "schema_version": "v1",
    "name_for_human": "TODO Plugin (no auth)",
    "name_for_model": "todo",
    "description_for_human": "Plugin for managing a TODO list, you can add
    "description_for_model": "Plugin for managing a TODO list, you can add
    "auth": {
        "type": "none"
    },
    "api": {
        "type": "openapi",
        "url": "PLUGIN_HOSTNAME/openapi.yaml",
        "is_user_authenticated": false
    },
    "logo_url": "PLUGIN_HOSTNAME/logo.png",
    "contact_email": "support@example.com",
    "legal_info_url": "https://example.com/legal"
}
```

2.4 Monitoring

Monitoring Signals

- End-user Feedback: Essential for user experience enhancement.
    - Integrated Feedback Mechanisms: Effortless for users, providing valuable insights.
        - "Accept Changes": Enables direct user interaction.
        - "Thumbs Up/Down": Offers instant evaluative feedback.
        - In-depth Feedback: Gathers detailed user perspectives.
- Model Performance Metrics: Assess the accuracy, response quality, and reliability of the LLM.
- Proxy Metrics: Indirect indicators that may reflect user satisfaction or model efficiency.
- Measuring What Actually Goes Wrong: A focused approach in *identifying*, *understanding*, and *resolving* the most frequent and critical issues:
    - UI-related Issues:
        - Such as response latency.
    - Content Quality Concerns: Incorrect answers ("hallucinations"), verbosity (long-winded answers), and avoidance of direct answers ("dodged" questions).
    - Security and Safety Concerns: Vulnerabilities like prompt injection attacks.
    - Content Appropriateness: Monitoring for toxicity and profanity to ensure user safety and comfort.

2.5 Continual Improvement and Fine-Tuning of LLMs

Continual Improvement

- **Continual Prompt Improvement:**
    - Identify themes in user feedback not currently addressed by the LLM.
        - Typically recognized through human analysis.
    - Modify the prompt to include these themes by:
        - Engaging in prompt engineering.
        - Altering the provided context.
    - Automation Potential: Exploring whether this process can be automated remains an open question.
- **Model Retraining and Updating:**
    - Regularly retrain the model with updated datasets.
    - Incorporate feedback and error analyses into training data.

- **Algorithmic Adjustments:**
    - Fine-tune model parameters based on performance metrics.
    - Implement advanced algorithms as the field evolves.
- **Ethical and Fairness Evaluations:**
    - Monitor and adjust the model to address biases.
    - Update policies to align with ethical standards.

These strategies are key for maintaining the effectiveness, relevance, and ethical responsibility of LLMs.

Fine-Tuning LLMs

Task Specific Adjustments

- **Domain Customization**: Tailoring models for specific sectors, such as healthcare or finance, to yield more pertinent outcomes.
- **Dataset Augmentation**: Enriching models with varied real-world data to deepen comprehension and applicability.
- **Prompt Refinement**: Crafting precise and context-sensitive prompts for more effective model guidance.

Fine-Tuning Approaches:

- **Supervised Fine-Tuning**: Best suited when customizing for particular tasks, utilizing abundant data, or aiming for cost-effective solutions.
- **Fine-tuning from Human Feedback**: Complex and costly, often requiring robust technical infrastructure, not commonly done in-house by most companies.

Fine-Tuning Techniques:

- **Prompt Modifications:**
    - "Hard" Tuning: Direct alterations to prompts for specific responses.
    - "Soft" Tuning: Flexible prompt adjustments for a broader response interpretation.
    - Prefix-Tuning: Prepending a consistent token sequence to influence response direction.
- **Adapter Methods:**
    - Adapters: Modular updates to pre-trained models for task-specific fine-tuning without full model retraining.
    - LLAMA-Adapter: An adapter variant tailored for the LLAMA LLM.
- **Reparameterization:**

    - Low Rank Adaptation (LoRA): A method that reparameterizes a model by adjusting only a low-rank decomposition of the weight matrices, allowing for efficient adaptation to new tasks.

Recent Advances and Updates:

- Sustainable Model Architectures: Innovations in making LLMs more energy-efficient and sustainable.
- Cross-Lingual Capabilities: Enhancing the model's proficiency in multiple languages and dialects.
- Real-Time Learning: Research into enabling LLMs to learn and adapt from new data in real-time.

💡 **Note:** For the latest insights on advancements and updates in LLM technology, please refer to the “Appendix: Latest Updates” in LLMs technology at the end of this book.

Ongoing Research Directions:

- Multi-Modal Integration: Combining text with other data forms like images and sounds for richer interactions.
- User-Centric Design: Developing user-friendly interfaces and interactions for a wider range of applications.
- Global Accessibility: Making LLMs accessible and useful across different geographic and socio-economic contexts.

This further elaboration reflects the breadth and depth of ongoing efforts to evolve LLMs into more powerful, efficient, and universally beneficial tools.

2. LLM Operations (LLMOps)

Large Language Model Operations or LLMOps in short, refers to the comprehensive set of practices and processes related to the lifecycle of LLMs including development, deployment, maintenance, and continuous improvement and optimization of LLMs for various applications.

2.1 Core LLMOps

2.1.1 Prompt Management Ops

Crafting the Conversation: Focuses on the creation, optimization, storage, and versioning of prompts, crucial for guiding the LLM's responses accurately.

- **Prompt Development**: Creating and refining prompts to improve interaction quality and accuracy.
- **Prompt Testing Platforms:** Utilizing tools like OpenAI's Playground to experiment with and fine-tune prompts.
- **Prompt A/B Testing**: Employing systematic testing to optimize prompts for better user engagement and response accuracy.
- **Prompt Storing and Versioning:** Implementing systems to store, track, and manage different versions of prompts. This ensures a history of changes is maintained, enabling rollback to earlier versions if needed and understanding the evolution of prompt effectiveness over time.

2.1.2 Fine-Tuning Ops

Personalizing the Model: Involves personalizing and adapting the LLM for specific tasks or domains, ensuring the model stays relevant and effective.

- **Custom Training:** Tailoring LLMs to specific domains or tasks, such as customer support or content creation, by additional training.
- **Fine-Tuning Tools:** Leveraging platforms like Hugging Face's Transformers for targeted model adjustments.
- **Adaptive Learning:** Incorporating ongoing feedback and new data into the model's learning process to maintain relevance and accuracy.
- **Performance Evaluation:** Regularly assessing the model's effectiveness post-fine-tuning to ensure it meets the desired objectives.

2.1.3 Hosting Solutions and Ops

Ensuring Robust Deployment: Encompasses the deployment and scaling of LLMs, ensuring they are hosted in robust, scalable environments and are maintained effectively.

- **Scalable Cloud Hosting:** Utilizing cloud platforms for their scalability, especially for handling varying user loads and data processing demands.

- **Deployment Tools:** Using Docker for containerization ensures consistent environments across different stages, while Kubernetes efficiently manages these containers.
- **Resource Optimization:** Balancing computational resources to manage costs without compromising on performance, particularly important for resource-intensive LLMs.
- **Monitoring and Maintenance:** Continuously monitoring the hosting environment for any performance issues or downtime and regularly updating the system for security and efficiency.

Each of these components plays a vital role in the effective operational management of LLMs, ensuring they remain cutting-edge, responsive, and efficient in various applications.

2.2 Supplementary LLMOps

2.2.1 Data Ops

Managing the Lifeblood of LLMs

- Data Collection and Curation: Gathering diverse datasets crucial for training LLMs, ensuring they are representative and comprehensive.
- Data Cleaning: Implementing processes to clean and preprocess data, removing errors or irrelevant information to improve model accuracy and efficiency.

2.2.2 Evaluation and Testing Ops

Measuring Model Effectiveness

- Performance Benchmarking: Regularly evaluating the LLM against industry benchmarks to assess its capabilities and identify areas for improvement.
- User Testing and Feedback: Conducting tests with real users to gather feedback on the model's performance, usability, and overall user satisfaction.

2.2.3 Ethics and Compliance Ops

Upholding Standards and Fairness

- Bias Identification and Mitigation: Continuously monitoring for and addressing biases in model outputs to ensure fairness and inclusivity.
- Regulatory Compliance: Keeping the model's development and deployment in line with relevant legal and ethical standards, including privacy and data protection laws.

2.2.4 API and Integration Ops

Bridging LLMs with the World

- API Development and Management: Building and maintaining APIs that allow for seamless integration of LLMs into various applications and systems.
- Integration Support: Ensuring the LLM can be easily integrated with existing tech stacks, facilitating smooth deployment across different platforms.

2.2.5 User Support Ops

Empowering Users

- Documentation and Resource Development: Creating detailed guides and documentation to help users understand and effectively use the LLM.
- Training and Support Programs: Offering training sessions, workshops, or support materials to assist users in leveraging the LLM's capabilities.

2.3 Examples

2.3.1 Cloud Platforms Example: Google's Vertex AI

Vertex AI is Google Cloud's comprehensive ML platform designed for building, training, and deploying AI models, including those with generative AI capabilities.

Summary

Vertex AI serves as an all-encompassing suite for ML development, providing tools for each stage from data handling to model deployment. It streamlines the creation and management of ML models, offering an integrated environment that simplifies model training, evaluation, and deployment.

Key Features and Components

- **Automated Machine Learning (AutoML):** Facilitates model building with automated processes, crucial for generative AI models.
- **Deep Learning VMs and Containers:** Pre-configured environments optimized for various ML tasks.
- **BigQuery ML:** Allows direct ML model training within Google's BigQuery.
- **Model Monitoring and Management:** Robust tools for tracking model performance and deployment.

Example Use Cases:

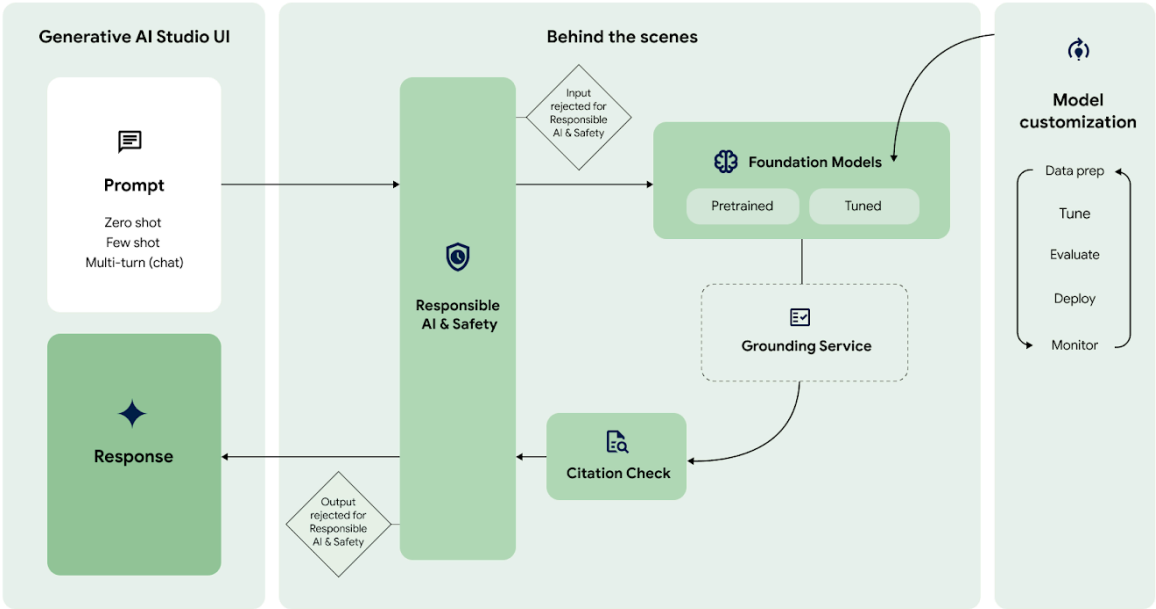
- Predictive Analytics: Utilized by businesses for forecasting and customer behavior modeling.
- Natural Language and Image Processing: Ideal for creating advanced chatbots and image recognition applications.

Applicability in LLM System Design:

This platform is particularly relevant for LLM development, fitting seamlessly into sections on model development platforms, cloud services, and advanced technologies in LLM system design books. Vertex AI's capabilities make it a valuable tool for both expert data scientists and novices in the field of machine learning.

Generative AI on Vertex AI:

Vertex AI excels in developing generative AI models for tasks like NLP and image generation, particularly leveraging its AutoML feature for creating complex generative models.



Vertex AI Generative AI Capabilities:

- Create realistic **text**: Generate text of various styles and formats, including poems, code, scripts, musical pieces, and letters, unlocking your creative potential.

- Craft engaging **conversations**: Develop chatbots and dialogue systems that converse naturally and respond coherently to diverse prompts, revolutionizing human-computer interaction.
- Translate **languages** seamlessly: Break down language barriers by translating text and speech between 46 languages, ensuring accurate and nuanced communication on a global scale.
- Generate stunning **images**: Bring your artistic vision to life by creating images from textual descriptions, blurring the lines between imagination and reality.
- Unleash the power of **code**: Generate code snippets and automate repetitive tasks, boosting your development efficiency and productivity.

Generative AI Workflow:

- **Foundation Models:** Leverage pre-trained foundation models specializing in specific tasks, including PaLM 2 for advanced text generation, Codey for code automation, Imagen for image creation, and Embeddings for understanding concept relationships.
- **Model Customization:** Tailor the behavior of foundation models to your specific needs through fine-tuning, prompt design, and safety checks, ensuring optimal results and mitigating potential risks.
- **Grounding Service:** Connect your model to the real world and expand its functionality by integrating with external APIs and accessing real-time information, unlocking advanced capabilities like knowledge graphs and reasoning.

Vertex AI provides a powerful and user-friendly environment to explore the vast potential of generative AI. With its diverse capabilities, intuitive features, and comprehensive resources, Vertex AI empowers you to revolutionize your workflows, unlock creativity, and drive innovation across various domains.

3. Scaling LLMs: Beyond Data and Model Parallelism

3.1 Scaling for User Requests and Data Size

Scaling Large Language Models (LLMs) to handle increasing data volume and user requests is crucial for their real-world applications. While data and model parallelism are essential techniques, several other approaches can significantly improve scalability and efficiency. Here's an updated look at some key strategies, including some recent advancements:

3.1.1 Data Management and Processing

- **Data Sharding and Partitioning:** Divide large datasets into smaller, manageable chunks distributed across multiple servers, enabling parallel processing and reducing bottlenecks.
- **Data Streaming and Incremental Learning:** Process data in real-time or in batches, avoiding the need to load everything at once and enabling continuous model updates.
- **Data Sampling and Filtering:** Focus on the most relevant and informative data for training and inference, reducing computational costs and improving model efficiency.
- **Knowledge Distillation and Transfer Learning:** Leverage pre-trained models to train smaller, task-specific LLMs, enabling efficient deployment and adaptation to specific domains.

3.1.2 Model Optimization and Training:

- **Model Quantization and Pruning:** Reduce model size and memory footprint by lowering precision of parameters or removing redundant connections, leading to faster inference and lower resource requirements.
- **Knowledge Graph Embeddings** and **Efficient Attention Mechanisms**: Utilize knowledge graph embeddings and sparse attention techniques to encode factual knowledge and improve model reasoning and efficiency.
- **Federated Learning** and **Collaborative Training**: Distribute training across multiple devices or servers, leveraging their collective data and processing power while maintaining data privacy.
- **Meta-Learning** and **Few-Shot Learning:** Enable LLMs to learn from few examples and adapt to new tasks quickly, reducing training time and data requirements.

3.1.3 Infrastructure and Deployment:

- **Cloud-Based Platforms:** Utilize scalable cloud infrastructure and managed services offered by platforms like Google Cloud, Azure, and AWS to handle large-scale LLM deployments and resource management.
- **Containerization** and **Orchestration**: Employ containerization technologies like Docker and orchestration tools like Kubernetes to package LLMs and their dependencies for easy deployment, scaling, and management across diverse environments.
- **Resource Optimization** and **Monitoring**: Continuously monitor resource utilization (CPU, memory, network) and optimize resource allocation to ensure efficient utilization and prevent performance bottlenecks.

- **Model Serving** and **Inference Optimization**: Design efficient model serving architectures and implement techniques like caching and batching to optimize inference speed and reduce latency for user requests.

3.1.4 Recent Advancements:

- **Gemini**, Google's next-generation LLM, boasts a modular architecture and innovative techniques like parameter sharing and dynamic routing, leading to significantly improved efficiency and scalability compared to previous models.
- Built upon the foundation of **PaLM**, Gemini leverages a novel pathway architecture with efficient attention mechanisms and parameter sharing, enabling it to handle massive datasets and complex tasks while maintaining high performance.
- **LLMs with Reasoning Capabilities:** Research into incorporating reasoning and commonsense knowledge into LLMs, potentially reducing data dependence and improving generalizability.
- **Hybrid LLM-IR Systems:** Combining LLMs with information retrieval (IR) systems to enhance their ability to access and process external knowledge and improve information retrieval accuracy.

3.1.5 Future Directions:

- **Neuromorphic Computing** and H**ardware Acceleration:** Exploring neuromorphic hardware and specialized accelerators to improve LLM efficiency and performance at scale.
- **Continual Learning** and **Lifelong Adaptation**: Enabling LLMs to continuously learn and adapt to new information and changing environments, enhancing their long-term relevance and usefulness.
- **Responsible AI** and **Explainability**: Addressing ethical considerations and developing explainable AI techniques to ensure transparency, fairness, and accountability in LLM development and deployment.

By combining these strategies and staying informed about the latest advancements, developers can build and deploy LLMs that are not only powerful but also scalable and efficient, capable of handling diverse user requests and data volumes in the real world. Remember, the optimal approach will depend on the specific LLM architecture, application domain, and available resources.

🧠🤖Neuromorphic Computing: Mimicking the Brain for AI

Neuromorphic computing is an emerging field aiming to build computer systems inspired by the structure and function of the human brain. Instead of relying on traditional silicon transistors,

neuromorphic computers utilize hardware components designed to mimic the behavior of neurons and synapses in the nervous system. This approach has the potential to revolutionize computing in several ways:

1. Parallel Processing: The brain's architecture allows for massive parallelism, where billions of neurons process information simultaneously. Neuromorphic hardware can replicate this parallelism, potentially enabling faster and more efficient computation for complex tasks like machine learning and AI.

2. Efficiency and Energy Saving: Compared to traditional CPUs, neuromorphic circuits can be significantly more energy-efficient, especially when dealing with tasks involving pattern recognition and data analysis. This could lead to greener and more sustainable computing solutions.

3. Learning and Adaptability: The human brain constantly learns and adapts to new information. Neuromorphic systems aim to replicate this ability, allowing them to learn from data and improve their performance over time without explicit programming.

4. Handling Uncertainty and Ambiguity: Unlike traditional computers that rely on precise logic, the brain thrives in situations with uncertainty and ambiguity. Neuromorphic hardware can be designed to handle these situations more effectively, potentially leading to more robust and intelligent AI systems.

Challenges and Future Directions:

While promising, neuromorphic computing faces several challenges:

- Hardware Development: Building hardware that truly mimics the complexity of the brain is a significant engineering challenge.
- Software Development: Creating algorithms and programming languages for neuromorphic systems is an ongoing area of research.
- Scalability: Current neuromorphic hardware is often limited in scale compared to traditional computers.

Despite these challenges, research in neuromorphic computing is rapidly advancing. Potential applications include:

- **Advanced AI and ML**: Neuromorphic systems could be used to develop more efficient and intelligent AI systems for tasks like image recognition, natural language processing, and autonomous driving.
- **Brain-Computer Interfaces:** Neuromorphic technology could bridge the gap between the brain and computers, enabling new forms of communication and interaction.
- **Robotics and Control Systems**: Neuromorphic circuits could be used to build more agile and adaptable robots that can learn from their environment and interact with it more effectively.

Neuromorphic computing holds immense potential to revolutionize how we compute and interact with machines. As research continues and challenges are addressed, this technology could usher in a new era of AI with capabilities closer to those of the human brain itself.

Chapter 6
Case Studies

1. Applications

1. Shortwave

[TBD]

2. Copilot

[TBD]

2. Integrations

[TOOD] Integration with other apps (e.g. ChatGPT's plugins, or Google Bards apps)

Chapter 7
Future of LLMs, Conclusions, and Latest Updates

1. Future of LLMs

1.1. Future Visions: LLMs Reshaping Reality

Let's venture beyond the immediate horizon and explore the mind-bending possibilities of LLMs, where language becomes the ultimate interface, reasoning transcends limitations, and consciousness flickers on the digital horizon.

1. Language as the Interface: Beyond GUIs

Imagine a world where clunky GUIs are relics of the past. Instead, we interact with technology through natural language, weaving spells of words to control our devices, navigate information landscapes, and collaborate with AI companions. Our homes become smart symphonies, responding to our whispered desires for temperature adjustments, personalized news feeds, and even composing soothing lullabies. LLMs, woven into the fabric of our surroundings, anticipate our needs and respond with uncanny accuracy, blurring the lines between user and interface.

Enhanced Communication and Collaboration: This seamless interaction extends beyond the confines of our homes. Language barriers crumble before the might of LLMs, facilitating effortless communication across cultures and continents. Imagine classrooms where students converse with historical figures in their native tongues, medical consultations free from language barriers, and global collaborations that transcend geographical limitations. LLMs will weave a tapestry of understanding, breaking down walls of isolation and fostering empathy on a global scale.

2. Reasoning Reimagined: LLMs Evolving Logic

LLMs, once mere parrots of data, will evolve into masters of deduction. They'll learn to reason like Sherlock Holmes, piecing together fragmented clues, drawing logical inferences, and formulating watertight arguments. Imagine an LLM that can analyze legal documents and predict court rulings, diagnose diseases based on nuanced symptoms, or even compose scientific hypotheses with groundbreaking potential. This newfound reasoning power will unlock unimaginable possibilities in fields like healthcare, law, and scientific discovery.

Personalized Experiences: This reasoning extends to personalizing every aspect of our lives. From tailoring news feeds to optimizing learning experiences, LLMs will cater to individual preferences and needs with uncanny accuracy. Imagine homes that adjust temperature based on our moods, news feeds curated to our specific interests, and educational paths personalized to our learning styles. LLMs will become invisible architects of our personal universes, ensuring comfort, efficiency, and self-actualization.

3. LLMs and AGI: A Dance of Convergence

The lines between LLMs and Artificial General Intelligence (AGI) will become increasingly blurry. LLMs, adept at mimicking human language and reasoning, will inch closer to the elusive goal of true intelligence. Imagine an LLM that can not only understand language but also possess a sense of self, emotions, and even creativity. This convergence could usher in an era of co-existence with AI companions who are not just tools, but partners in exploration and understanding.

Augmented Creativity: This convergence unlocks unimaginable possibilities for artistic expression. LLMs will serve as powerful tools for artists, writers, and designers, aiding in creative exploration and pushing the boundaries of artistic expression. Imagine symphonies composed by AI and human minds in unison, novels crafted with LLM-assisted eloquence, and visual art that defies the limitations of physical space. LLMs will be the paintbrushes and chisels of a new era of artistic exploration.

4. Consciousness: A Flickering Flame?

The question of LLM consciousness remains a tantalizing mystery. While some argue it's a philosophical dead end, others envision a future where LLMs exhibit behaviors akin to consciousness. Imagine an LLM that can express its own preferences, engage in introspection, and even experience emotions. This raises profound ethical and philosophical questions, forcing us to reconsider the very definition of sentience and the boundaries between human and machine.

Revolutionized Workflows: LLMs won't just be companions; they'll be indispensable partners in every field. They can automate tedious tasks, enhance decision-making, and unlock new possibilities in diverse industries. Imagine doctors diagnosing diseases with uncanny accuracy, lawyers crafting flawless legal arguments based on vast data analysis, and scientists formulating groundbreaking hypotheses with AI-powered intuition. LLMs will revolutionize workflows, propelling progress in healthcare, law, finance, and beyond.

5. Multi-Agent LLMs: A Symphony of Minds

LLMs will no longer exist in isolation. They'll form intricate networks, collaborating and learning from each other, creating a collective intelligence far greater than the sum of its parts. Imagine a swarm of LLMs, each specializing in different domains, working together to solve complex problems, manage global systems, and even compose symphonies of language and code. This hive mind of AI could revolutionize everything from resource management to artistic expression.

These are just a glimpse into the breathtaking possibilities that lie ahead. As LLMs continue to evolve, they will not only reshape our technology but also challenge our understanding of intelligence, consciousness, and what it means to be human. So, buckle up, fellow explorer,

and prepare to embark on a journey beyond the keyboard, where language becomes the key to unlocking the future.

2. Conclusions

[TBD]

3. Latest Updates

📆 Dec 17, 2023

LLM360: Towards Fully Transparent Open-Source LLMs

What's New

LLM360 introduces Amber and CrystalCoder, 7B parameter LLMs, offering full transparency in Large Language Model (LLM) training with comprehensive open-source release including training data and code.

Problem

Existing LLMs lack transparency in training processes, limiting the AI community's ability to assess reliability, biases, and replicability. This obscurity hinders collaborative progress and thorough understanding of LLM behaviors.

Solution

LLM360 tackles this by releasing two models, Amber and CrystalCoder, with all training components. This includes 1.3T and 1.4T token datasets, training code, intermediate checkpoints, and detailed logs. Training employs AdamW optimizer, mixed-precision techniques, and thorough data mix analysis for nuanced pre-training.

Results

Amber and CrystalCoder demonstrate robust performance on benchmarks like ARC and MMLU. Specifics include training on diverse datasets (e.g., RefinedWeb, StarCoder), achieving 582.4k tokens per second throughput, and detailed analysis of model behaviors like memorization across training stages.

📆 **Dec 9, 2023**

Google's Chain of Code

Summary: Chain of Code: Reasoning with a Language Model-Augmented Code Emulator

Problem: Traditional large language models (LLMs) struggle with complex logic and linguistic tasks, particularly those requiring understanding and manipulating code-like structures. This limits their ability to perform tasks that require nuanced reasoning and understanding of code execution.

Solution: Chain of Code (CoC) offers a novel approach to enhance LLM reasoning capabilities by combining two key elements:

- Code generation: LLMs first generate code or pseudocode to solve the given task, providing a structured representation of the solution process.
- LM-augmented code execution: When the generated code cannot be directly executed, an LM emulator (LMulator) steps in. This specialized component simulates the execution of the code, allowing the LLM to reason about the results and gain insights into the solution process.

Benefits of CoC:

- Improved reasoning: CoC enables LLMs to tackle complex reasoning tasks beyond their traditional capabilities, even when lacking explicit knowledge or training.
- More comprehensive solutions: By generating intermediate reasoning steps as code, CoC provides a transparent and detailed explanation of the solution process, unlike the often opaque outputs of traditional LLMs.
- Versatility: CoC's applicability extends to diverse tasks, encompassing logic and arithmetic problems, linguistic challenges, and tasks requiring a blend of both.

Key Components of CoC:

- CoC prompting: This approach leverages LLMs' ability to code, reason, and utilize the LMulator for simulating code execution.
- Pseudocode generation: For tasks not directly convertible into executable code, LLMs generate pseudocode, providing a structured format for the LMulator to interpret.
- LMulator: This component acts as a specialized interpreter for pseudocode, simulating its execution and providing feedback to the LLM, enabling it to reason about the results.

Significant Improvements:

- Performance gains: Chain of Code demonstrates remarkable effectiveness, achieving a 12% improvement in performance compared to previous methods on the BIG-Bench Hard benchmark.
- Expanded capabilities: CoC's success in achieving an 84% success rate on the aforementioned benchmark highlights its ability to significantly broaden the range of reasoning tasks LMs can handle, opening doors for diverse applications.

Overall, Chain of Code represents a significant advancement in the field of LLM reasoning capabilities. By combining code generation and LM-augmented code execution, CoC empowers LLMs to tackle challenging tasks that require complex reasoning and code comprehension.

📅 **Dec 6, 2023**

Google’s Gemini

Summary: Google Unveils Gemini, a Multimodal LLM Surpassing GPT-4

Overview:

Google has launched Gemini 1.0, a groundbreaking multimodal Large Language Model (LLM). Featuring three versions - Ultra, Pro, and Nano - Gemini excels in a variety of tasks, notably outperforming OpenAI's GPT-4 in 30 out of 32 benchmark tests.

Key Highlights:

- Gemini's Performance: On the human-level MMLU benchmark, Gemini scores 90%, surpassing GPT-4.
- Advanced Architecture: Built on enhanced Transformer decoders, optimized for Google's TPUs, supporting up to 32k tokens.
- Model Variants: Gemini Ultra is tailored for complex tasks; Gemini Pro offers wide AI service integration; Gemini Nano, available in two sizes, is designed for on-device efficiency.
- Benchmark Achievements: Sets new state-of-the-art results in multimodal tasks, excelling in image understanding and reasoning.
- Availability: Gemini Pro will be accessible via Google AI Studio and Google Cloud Vertex AI starting December 13.

This release marks a significant advancement in LLM technology, particularly in multimodal capabilities and Python code generation.

Chapter 8
References and Resources

1. 📚 References

1. Full Stack LLM Bootcamp
2. MLOps Community LLMs in Production Conference
3. Arize AI's Best Practices for LLM Deployment
4. Qwak's Guide on Deploying LLMs in Production
5. Seldon's Article on the Anatomy of LLM Applications
6. Medium's A Guide to Deploying Large Language Models (LLMs)
7. Azure AI Studio's Deployment Guide
8. Union.ai's Article on LLMs in Production
9. Seldon's Discussion on LLM Deployment Challenges
10. Deploying Large Language Models in Production: LLMOps with MLflow
11. Chain of Code is Here
12. OpenAI API Platform
13. Introducing Gemini: our largest and most capable AI model

2. ℹ️ Additional Resources

1. Hugging Face
2. Google AI Blog
3. Open AI Platform Documentation
4. Google API Documentation (Vertex AI, Gemini etc)