Advanced Topics and Future Directions Module

Advanced Topics and Future Directions

Goals: Equip students with an understanding of the latest trends in LLM.

This section examines the nuances of multimodal LLMs, scaling laws in training, sophisticated prompting frameworks like **ReAct**, and the complexities of training on generated data. The participants will unravel the ways they redefine interactions with language models. **FlashAttention**, **Sparse Attention**, and **ALiBi's** roles in expanding the context window offer insights into LLM optimization strategies. The module also confronts the challenges of training on generated data, spotlighting the critical phenomenon of Model Collapse.

- **Multimodal LLMs:** Discover the significance of multimodality in LLMs, integration of varied data like text and images. The lesson highlights popular models adept at multimodality and provides an overview of their functionalities.
- **Scaling Laws in LLM Training**: To achieve compute-optimal training results, it's essential to maintain a harmonious balance between the size of the LLM and the number of training tokens. This session offers guidance on how to effectively scale both elements in tandem.
- **ReAct framework and ChatGPT plugins**: The ReAct framework offers a sophisticated approach to enhancing interactions with language models, which also abilitates ChatGPT plugins.
- **Expanding the context window**: Detailed workings of FlashAttention and Sparse Attention in the transformer structures, aiming for efficient computation. Emphasis is also given to ALiBi's significance in this context.
- **Training on generated data: Model Collapse:** The problem of training on generated data, particularly emphasizing the phenomenon of Model Collapse. Understanding this topic is crucial, as training on bad data influences the effectiveness and reliability of LLMs.
- **New Challenges in LLM Research**: The Emerging challenges in LLM research, are cover agents, retriever architectures, larger context windows, efficient attention, and cost-effective pre-training and fine-tuning. Insights from recent studies guide the exploration, preparing students for the evolving LLM landscape.

As we wrap up this comprehensive course on Training and Fine-Tuning Large Language Models, students have covered a vast spectrum of topics, from the basic architecture of LLMs to deployment, from specialized Fine-Tuning methods to anticipated challenges in LLM research. With this knowledge, students are well-prepared to apply LLMs in practical scenarios, evaluate their performance effectively, innovate with fine-tuning strategies, and stay abreast of emerging trends and challenges in the domain of artificial intelligence and machine learning.

Introduction to Large Multimodal Models

Introduction

In this lesson, we will examine an emerging field of interest: The next progression in the evolution of LLMs, **Large multimodal models (LMMs)**. This topic adds a new layer to the material we've covered throughout the course.

Simply put, multimodal models are designed to handle and interpret different data types - or **"modalities"** - such as **text, images, audio, and video**, all within a single coordinated system. This integration allows for a more comprehensive analysis and understanding than models processing only one data type, such as text in standard LLMs. For instance, supplementing a text prompt with voice or image inputs can enable these models to capture a more complex representation of the conveyed information. This is achieved by analyzing additional layers of data, such as the tone and cadence of your voice or the visual context provided by images, thus enhancing the depth and richness of the analysis.

With the recent increase in the popularity of large language models, it is unsurprising that researchers are now exploring the potential of extending these models to handle multiple data types, aiming to create more versatile and valuable **general-purpose assistants**. Models that can solve arbitrary tasks specified by the user.

In the following sections, we will explore the current implementations of LMMs and introduce key concepts on how they manage multimodality. We will also learn about their **emergent abilities** and explore the idea of **Instruction-tuned** LMMs.

Finally, in this lesson, we will learn how **Deep Lake** by Activeloop can be helpful in training or fine-tuning large multimodal models.

Common Architectures and Training Objectives

By definition, multimodal models are designed to process various **input modalities, such as text, images,** and **videos,** and **generate outputs in multiple modalities**. However, a notable subset of currently popular LMMs primarily focuses on **accepting image inputs** and **is limited to generating text outputs**.

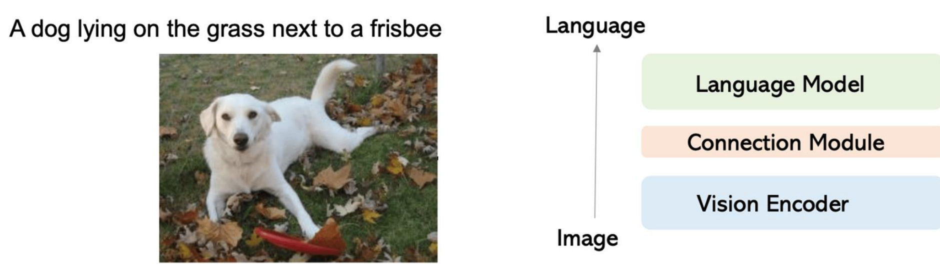
These specialized LMMs often leverage pre-trained large-scale vision or language models as their foundation. We can categorize them as **'Image-to-Text Generative Models,'** also known as **visual language models (VLMs).** They generally perform tasks related to image understanding, such as question answering and image captioning. Examples include GIT by Microsoft, BLIP2 by SalesForce, and Flamingo by DeepMind.

Model Architecture

These models use an **image encoder** to extract visual features and a standard LLM to output a text sequence. The image encoder can be based on convolutional neural networks (**CNNs**), such as ResNet, or a transformer-based architecture like the **Vision Transformer (ViT).**

The image encoder and the language model can be trained from scratch or using pre-trained models. Most state-of-the-art models opt for the latter approach; an example is the **pre-trained image encoder** from the model **CLIP** by **OpenAI.** The options for language models are also extensive: one could choose from various open-source pre-trained models, such as Meta's OPT, Llama 2, or Google's instruction-trained FlanT5 models.

Optionally, models like BLIP2 introduce a trainable lightweight connection module connecting the vision and language modalities. Since BLIP2 only trains this light module, it is cheaper and faster than other methods while still managing a **strong zero-shot performance** on image understanding tasks.



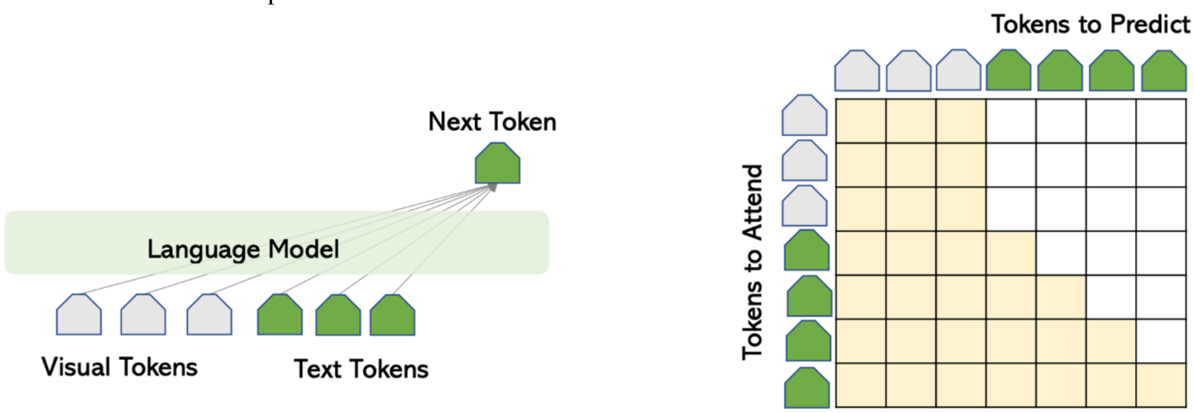
(a) Left: An example of image-to-text generation task; Right: model architecture.

From "Multimodal Foundation Models: From Specialists to General-Purpose Assistants" paper

Training Objective

Similar to what we've seen in the course, **LMMs** are trained using an **auto-regressive loss** function applied to the output text tokens. When using a Vision Transformer architecture, the concept of '**image tokens**,' which is analogous to text tokenization, is introduced. Just like text can be divided into smaller units like sentences, words, or sub-words for easier processing, **images can be segmented into smaller, non-overlapping patches, known as 'tokens.'**

The exact attention mechanisms come into play in the Transformer architecture employed by these LMMs. **Image tokens can 'attend' to each other**, meaning they can influence each other's representation in the model. Meanwhile, the generation of each text token depends on all the image tokens and the previously generated text tokens. Check out our lesson about **Understanding Transformers** if you are still getting familiar with these concepts.



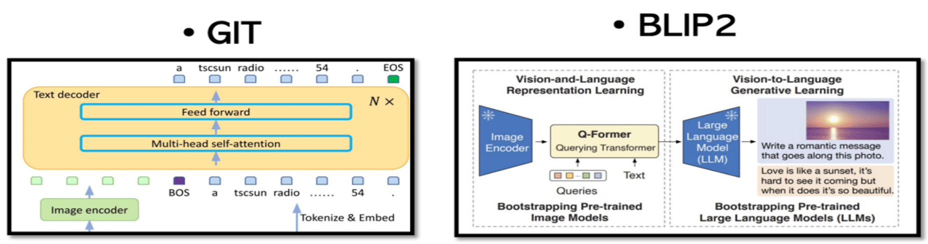
(b) Training objective and attention mask. For each row, the yellow elements indicate that the prediction token attends the tokens on the left.

Figure 3: Illustration of image-to-text generation task, architecture, and training objective.

From "Multimodal Foundation Models: From Specialists to General-Purpose Assistants" paper

Differences in Training Schemes

While having the same training objective, some variations emerge in the training schemes of different Language-Multimodal Models (LMMs). Most models, such as **GIT** and **BLIP2**, employ **only image-text pairs for training**. This approach allows them to establish connections between the text and image representations effectively but requires a large, curated image-text pairs dataset.



	GIT	BLIP2
Language Model	From Scratch	Pre-trained: FLAN-T5/OPT
Connection Module		Q-Former: Lightweight Querying Transformer
Vision Encoder	Contrastive pre-trained: Florence/CLIP	Contrastive pre-trained: EVA/CLIP

(a) Example 1: LMM with Image-Text Pairs.

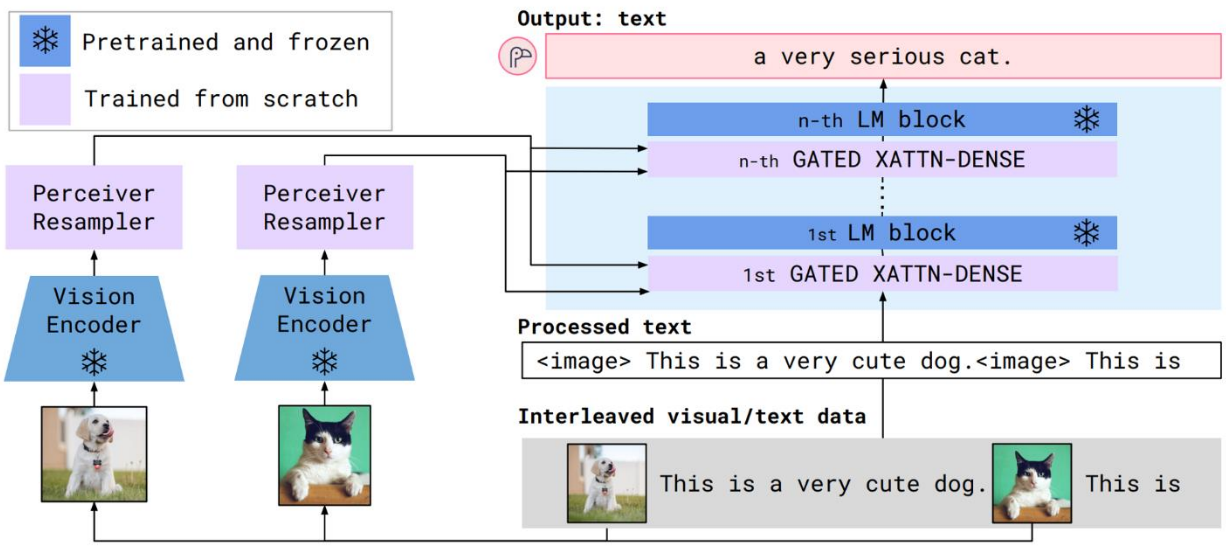
From "Multimodal Foundation Models: From Specialists to General-Purpose Assistants" paper

On the other hand, the **Flamingo** model has some architectural innovations that allow for unlabeled web training data. They extract the text and images from the HTML of 43M webpages. They also determine the positions of images relative to the text based on the relative positions of the text and image elements in the Document Object Model (DOM).

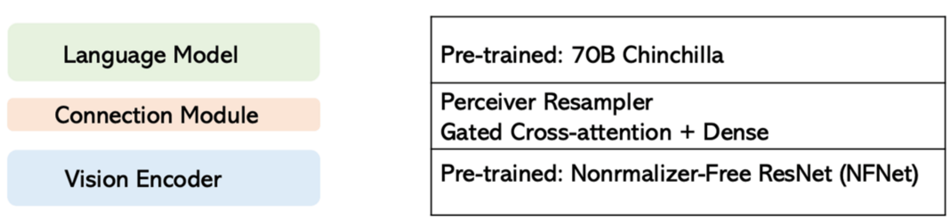
This model can ingest a **multimodal prompt** containing images and/or videos interleaved with text as input and generate text in an open-ended manner. It can produce text for tasks such as image captioning or visual question-answering.

The system connects the different modalities and enables multimodal prompting through steps. Initially, a **Perceiver Resampler** module receives spatiotemporal features from visual data, such as an image or video, processed by the pre-trained Vision Encoder. The Perceiver then generates fixed number of '**visual tokens**.' These visual tokens serve as inputs to condition a **frozen language model**, a pre-trained language model that is not updated during this process. The conditioning is facilitated by adding newly initialized **cross-attention layers** interleaved with the language model's pre-existing layers. These new layers are not frozen and will be updated during training.

While this architecture is less efficient by having more parameters to train than the one from BLIP2, it provides a powerful **way for the language model to incorporate visual cues**.



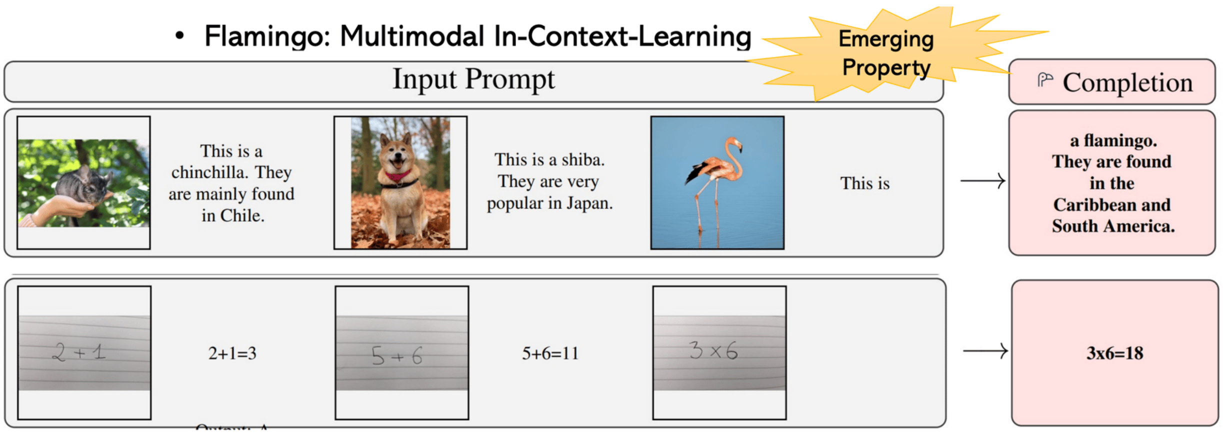
From "[Flamingo: a Visual Language Model for Few-Shot Learning]" paper



From "Multimodal Foundation Models: From Specialists to General-Purpose Assistants" paper

Discovering Emergent Abilities - Few-shot In-Context-Learning

Its flexible architecture allows Flamingo to be trained with multimodal prompts that interleave text with visual tokens. This enables the model to demonstrate emergent abilities, such as few-shot in-context learning, **analogous to GPT-3**. You can see some examples in the figure below.



From [“Multimodal Foundation Models: From Specialists to General-Purpose Assistants”]() paper

Open-sourcing Flamingo

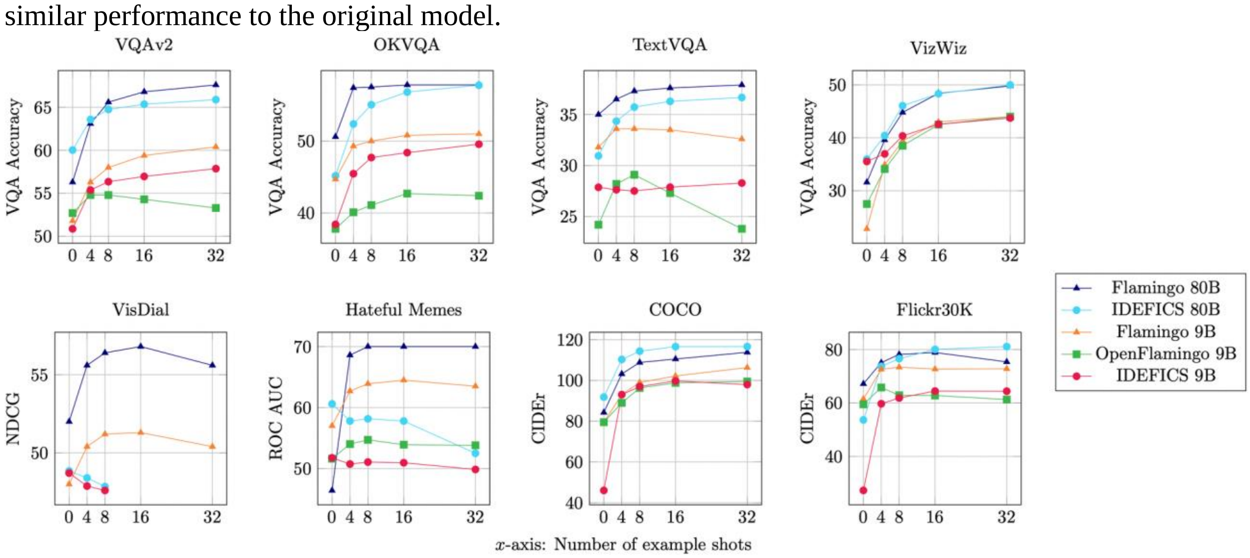
The state-of-the-art results reported in the Flamingo paper are exciting and clearly show significant progress in the field of LMMs. However, DeepMind has yet to make the Flamingo model publicly available.

To fill this gap, HuggingFace's team took the initiative to create an open-source reproduction of Flamingo, known as **IDEFICS**. This replica is constructed entirely using publicly accessible resources, including the LLaMA v1 and OpenCLIP models. IDEFICS is offered in the **'base'** and the **'instructed'** variants. Both of these are available in two sizes—**9 billion** parameters and **80 billion** parameters. IDEFICS offers comparable results to Flamingo.

The team used a mixture of openly available datasets such as Wikipedia, Public Multimodal Dataset, and LAION to train these models. They also created a new 115B token **dataset** called **OBELICS**. It has 141 million interleaved image-text documents scraped from the web and contains 353 million images, replicating the dataset described by DeepMind in the Flamingo paper.

IDEFICS is available through the Transformers library, and a demo of it is available here.

Another open-source replication of Flamingo is called **Open Flamingo** at the 9B parameter size, offering similar performance to the original model.



From “**Introducing IDEFICS: An Open Reproduction of State-of-the-Art Visual Language Model**” blog post.

Instruction-tuned LMMs

As demonstrated by GPT-3's emergent abilities with few-shot prompting, where the model could tackle tasks it hadn't seen during training, there's been a rising interest in instruction-fine-tuned LMMs. By allowing the models to be instruction-tuned, we can expect these models to perform a broader set of tasks and better alignment with human intents. This is line with the work done by OpenAI with InstructGPT and, more recently, GPT-4.

OpenAI has showcased the capability of their newer "**GPT-4 with vision**" model to follow instructions using visual inputs in their GPT4 technical report and GPT-4V(ision) System Card.

GPT-4 visual input example, Extreme Ironing:

User What is unusual about this image?

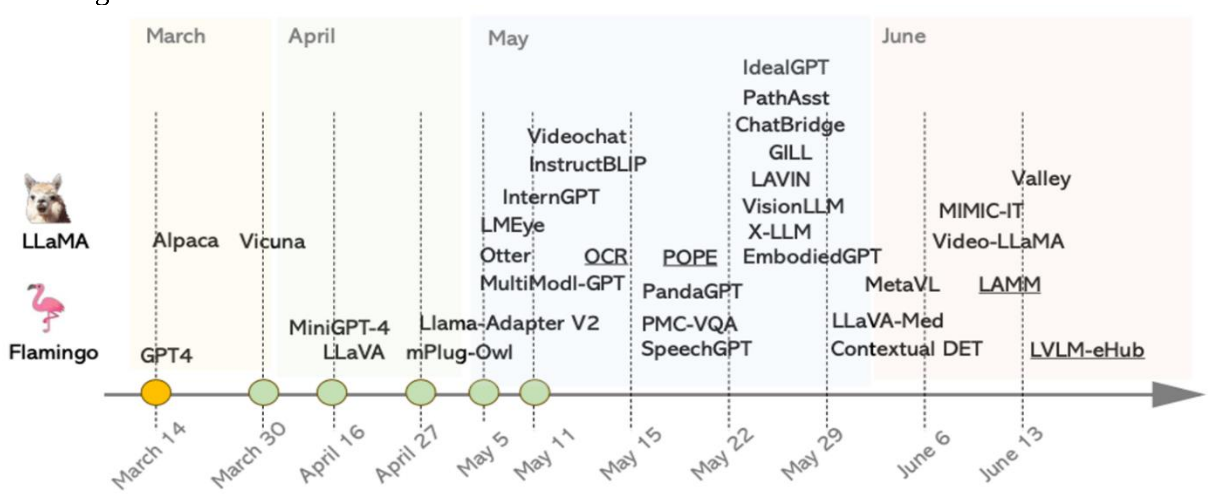
Source: https://www.barnorama.com/wp-content/uploads/2016/12/03-Confusing-Pictures.jpg

GPT-4 The unusual thing about this image is that a man is ironing clothes on an ironing board attached to the roof of a moving taxi.

Table 16. Example prompt demonstrating GPT-4's visual input capability. The prompt requires image understanding.

From the "GPT-4 Technical Report"

Following the announcement of OpenAI's **multimodal GPT-4**, there has been a surge in related research. As a result, multiple research labs have introduced instruction-tuned LMMs, including **LLaVA**, **MiniGPT-4**, and **InstructBlip**. They feature similar network architectures as previous LMMs but train on instruction-following datasets.



From "Multimodal Foundation Models: From Specialists to General-Purpose Assistants" paper

Exploring LLaVA - an instruction-tuned LMM

The network architecture of LLaVA resembles the one we reviewed before. This model connects a pre-trained CLIP visual encoder and the Vicuna language model **via a projection matrix**. In other words, they consider a simple linear layer to connect image features into the word embedding space. Specifically, they apply a trainable projection matrix called W to convert the image features into language embedding tokens with the same dimensionality as the word embedding space in the language model.

The authors of LLaVA chose these new linear projection layers that are more lightweight than the Q-Former connection module we saw for BLIP2 and the Perceiver Resampler and cross-attention layers from Flamingo.

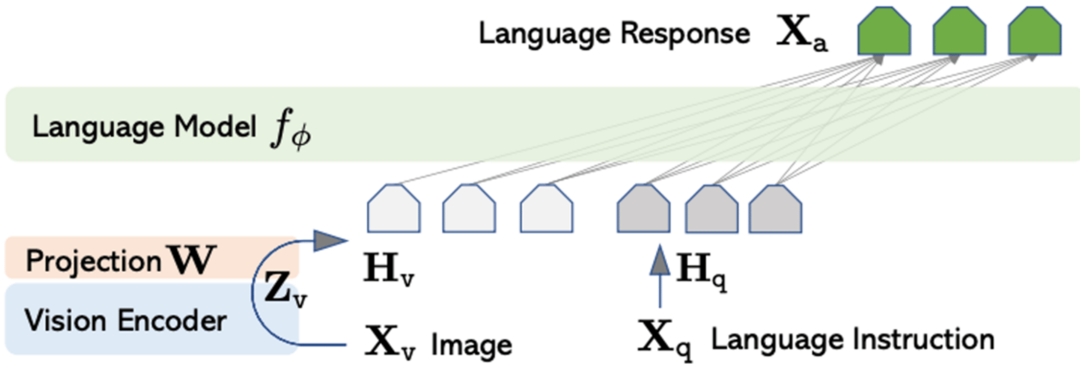


Figure 1: LLaVA network architecture.

From "Visual Instruction Tuning" paper

The authors then adopt a **two-stage instruction-tuning** procedure to train the model. First, they pre-train the projection matrix using a subset of the CC3M dataset, consisting of images and captions. Then, the model is finetuned end-to-end. Both the projection matrix and the LLM are trained on the proposed multimodal instruction-following data for daily user-oriented applications.

They also leverage GPT-4 to create a **synthetic dataset** consisting of multimodal instructions, drawing from widely available image-pair data.

In the dataset construction process, GPT-4 is shown symbolic representations of images using **captions** and the **coordinates of bounding boxes**, as depicted in the figure below. These representations are derived from the COCO dataset. This information is fed into GPT-4 as a prompt to generate training samples. The generated samples fall into **three categories**: question-answer conversations, detailed descriptions, and complex reasoning questions and answers.

They create 158K training samples in total.

Context type 1: Captions

A group of people standing outside of a black vehicle with various luggage.
Luggage surrounds a vehicle in an underground parking area
People try to fit all of their luggage in an SUV.
The sport utility vehicle is parked in the public garage, being packed for a trip
Some people with luggage near a van that is transporting it.

Context type 2: Boxes

person: [0.681, 0.242, 0.774, 0.694], person: [0.63, 0.222, 0.686, 0.516], person: [0.444, 0.233, 0.487, 0.34], backpack: [0.384, 0.696, 0.485, 0.914], backpack: [0.755, 0.413, 0.846, 0.692], suitcase: [0.758, 0.413, 0.845, 0.69], suitcase: [0.1, 0.497, 0.173, 0.579], bicycle: [0.282, 0.363, 0.327, 0.442], car: [0.786, 0.25, 0.848, 0.322], car: [0.783, 0.27, 0.827, 0.335], car: [0.86, 0.254, 0.891, 0.3], car: [0.261, 0.101, 0.787, 0.626]

Response type 1: conversation

Question: What type of vehicle is featured in the image?
Answer: The image features a black sport utility vehicle (SUV).
Question: Where is the vehicle parked?
Answer: The vehicle is parked in an underground parking area, likely in a public garage.
Question: What are the people in the image doing?
Answer: The people are trying to fit all of their luggage into the SUV, likely preparing for a trip.

Response type 2: detailed description

The image is an underground parking area with a black sport utility vehicle (SUV) parked. There are three people in the scene, with one person standing closer to the left side of the vehicle, another person in the middle, and the third person on the right side. They are all working together to pack their luggage into the SUV for a trip.
In the parking area, various luggage items are scattered around the vehicle. There are two backpacks, one located near the left rear wheel and the other closer to the right side of the vehicle. Additionally, there are two suitcases, one on the right side of the car and another further away near the center of the parking area. A bicycle can also be seen on the left side of the vehicle.
Other cars are parked around the main SUV, with one car positioned behind it and slightly to the left, another behind and slightly to the right, and the third car further behind on the right side.

Response type 3: complex reasoning

Question: What challenges do these people face?
Answer: In the image, a group of people is standing outside a black SUV in a parking area, surrounded by various pieces of luggage, including suitcases and backpacks. They are facing the challenge of fitting all their luggage into the black SUV. There are multiple suitcases and backpacks to be packed, which suggests that the group has a significant amount of belongings to accommodate. They might have to strategize and arrange the luggage efficiently to ensure that everything fits properly into the vehicle. Additionally, they need to consider the comfort of the passengers and visibility while driving, so the placement of the luggage must not obstruct the driver's view or make the passengers uncomfortable during the trip.

Table 1: One example to illustrate the instruction-following data. The top block shows the contexts such as captions and boxes used to prompt GPT, and the bottom block shows the three types of responses. Note that the visual image is not used to prompt GPT, we only show it here as a reference.

From "Visual Instruction Tuning" paper

The LLaVA model demonstrates the effectiveness of **visual instruction tuning using language-only GPT-4**. They show its capabilities by prompting the model with the same question and image as in the GPT-4 report. You can see the result below. The authors also report a new SOTA by fine-tuning ScienceQA, a benchmark that contains 21k multimodal multiple-choice questions with rich domain diversity across three subjects, 26 topics, 127 categories, and 379 skills.

Visual input example, Extreme Ironing:

Source: https://www.barnorama.com/wp-content/uploads/2016/12/03-Confusing-Pictures.jpg

User: What is unusual about this image?

LLaVA: The unusual aspect of this image is a man ironing clothes on the back of a minivan or van. This is not a typical place to perform this activity, as one would usually iron clothes in a more stationary and safe location, such as a home, using a regular ironing board. The scene depicted in the image is peculiar as it involves a makeshift ironing setup on a vehicle, which can be both unsafe and unconventional. Additionally, it is not clear how the man is able to maintain balance and stability while ironing clothes in such an unstable environment.

From "Visual Instruction Tuning" paper

Beyond vision and language

In recent months, Image-to-text generative models have dominated the Large Multimodal Model (LMM) landscape. However, newer models have emerged that embrace a wider range of modalities beyond just vision and language.

For instance, **PandaGPT** is designed to handle any input data type, thanks to its integration with the **ImageBind** encoder.

There's also **SpeechGPT**, a model that integrates text and speech data and generates speech alongside text.

NExT-GPT stands out as a versatile model capable of receiving and producing outputs in any modality."

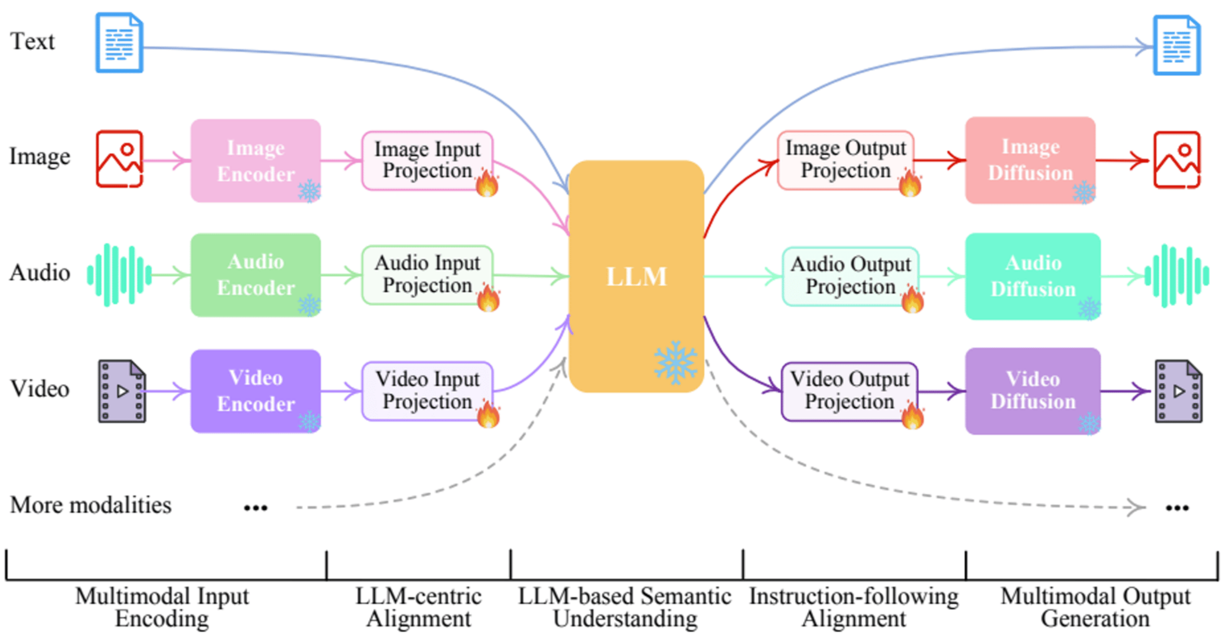
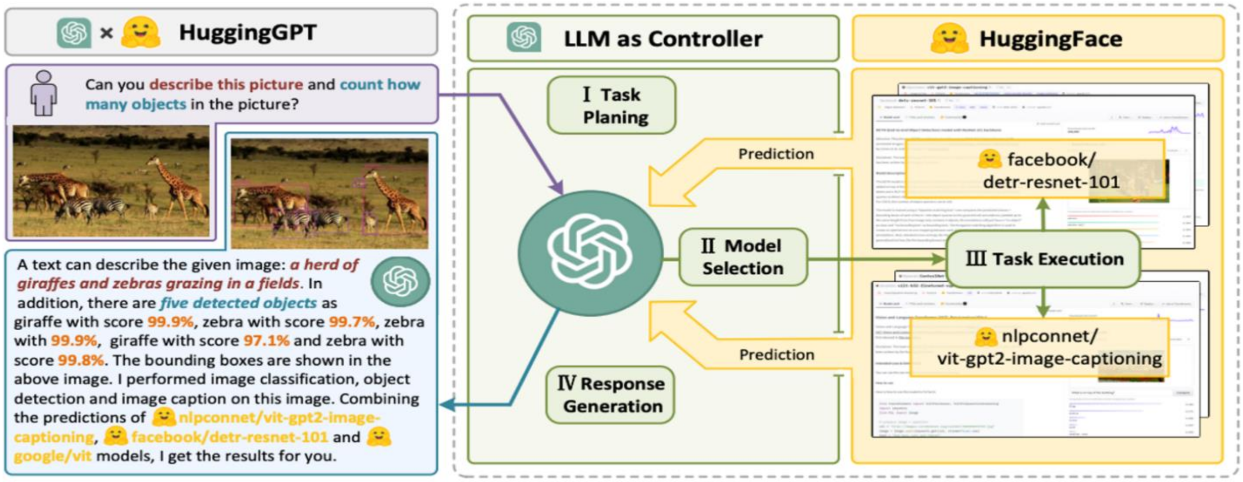


Figure 1: By connecting LLM with multimodal adaptors and diffusion decoders, NExT-GPT achieves universal multimodal understanding and any-to-any modality input and output.

From "NExT-GPT: Any-to-Any Multimodal LLM" paper

[HuggingGPT](#) is a novel system that integrates with the HuggingFace platform. It employs a Large Language Model (LLM) as its central controller. This LLM determines which specific model on HuggingFace is best suited for a task, selects that model, and then returns the model's output.



From “[HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face](#)” paper

Deep Lake and Multimodal LLMs

Deep Lake, differently from many other data lake and vector store products, is multi-modal and can store any data, from texts to images, videos, or audio. If you’re interested in building multi-modal LLMs, this is something to consider, as you can store the different types of data you need in the same place. See this [code example](#) to see how to store images and texts in the same dataset. Moreover, here’s the [full list of data types](#) that can be managed with Deep Lake.

HTYPE	DTYPE	COMPRESSION
generic	None	None
image	uint8	Required arg
image.rgb	uint8	Required arg
image.gray	uint8	Required arg
video	uint8	Required arg
audio	float64	Required arg
class_label	uint32	None
bbox	float32	None
bbox.3d	float32	None
intrinsics	float32	None
segment_mask	uint32	None
binary_mask	bool	None
keypoints_coco	int32	None
point	int32	None
polygon	float32	None
text	str	None
json	Any	None
list	List	None
dicom	None	dcm
nifti	None	Required arg
point_cloud	None	las
mesh	None	ply
instance_label	uint32	None
embedding	None	None
link	str	None
sequence	None	None

Conclusion

In this module, we delved into the emergent field of LMMs. We examined the leading models that combine both vision and language modalities. We learned that instruction-tuning allows these models to achieve greater generalization on tasks they haven't encountered before. Furthermore, we were introduced to advanced LMMs capable of integrating an even wider range of modalities. Lastly, we discussed the utility of Deep Lake in fine-tuning LMMs.

Scaling Laws in LLM Training

Introduction

In this lesson, we will study the relations between language model performance and parameters like model scale, model shape, and compute budget. The lesson is a small summary of extracts from the papers "Scaling Laws for Neural Language Models" and "Training Compute-Optimal Large Language Models."

A study on language modeling performance

The paper Scaling Laws for Neural Language Models (2020) contains a study of empirical scaling laws for language model performance on the cross-entropy loss, focusing on the Transformer architecture.

The experiments show the test loss scales as a power-law with model size, dataset size, and the amount of compute used for training; with some trends spanning more than seven orders of magnitude. This means simple equations govern the relationships between these variables, and these equations can be used to create an optimally efficient training configuration for training a very large language model. Moreover, it looks like other architectural details, such as network width or depth have minimal effects within a wide range.

As deduced from the experiments and the derived equations, larger models are significantly more sample efficient, i.e., optimally compute-efficient training involves training very large models on a relatively modest amount of data and stopping significantly before convergence.

Experiments

To study language model scaling, a variety of models have been trained with different factors, including:

- **Model size (*N*)**: ranging in size from 768 to 1.5 billion non-embedding parameters.
- **Dataset size (*D*)**: ranging from 22 million to 23 billion tokens.
- **Model shape**: including depth, width, attention heads, and feed-forward dimension.
- **Context length**: 1024 for most runs, with some experiments with shorter contexts.
- **Batch size**: 2^19 for most runs, with some variations to measure the critical batch size. Training at the critical batch size provides a roughly optimal compromise between time and compute efficiency.

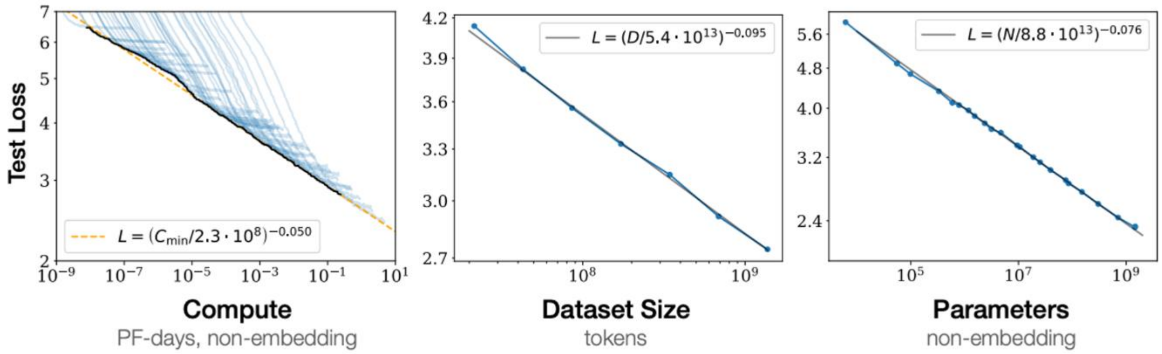
Let's define the following training variables as well:

- Let ***L*** be the **test cross-entropy loss**.
- Let ***C*** be the **amount of compute** used to train a model.

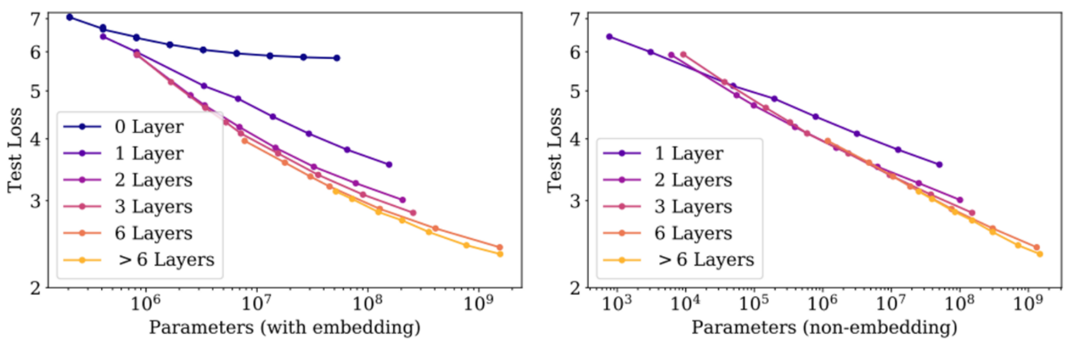
Key findings

Taking inspiration from section 1.1 of the paper, we summarize the results of the experiments.

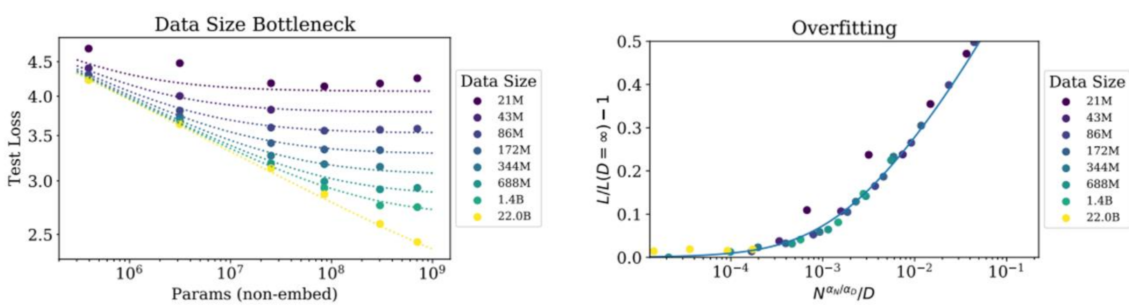
- **Performance depends strongly on model scale, weakly on model shape:** Model performance depends most strongly on scale, which consists of three factors: the number of **model parameters *N*** (excluding embeddings), the **size of the dataset *D***, and the **amount of compute *C*** used for training. Within reasonable limits, performance depends very weakly on other architectural hyperparameters, such as depth vs. width.
- **Smooth power laws**: Performance has a power-law relationship with each of the three scale factors *N*, *D*, and *C* when not bottlenecked by the other two, with trends spanning more than six orders of magnitude.



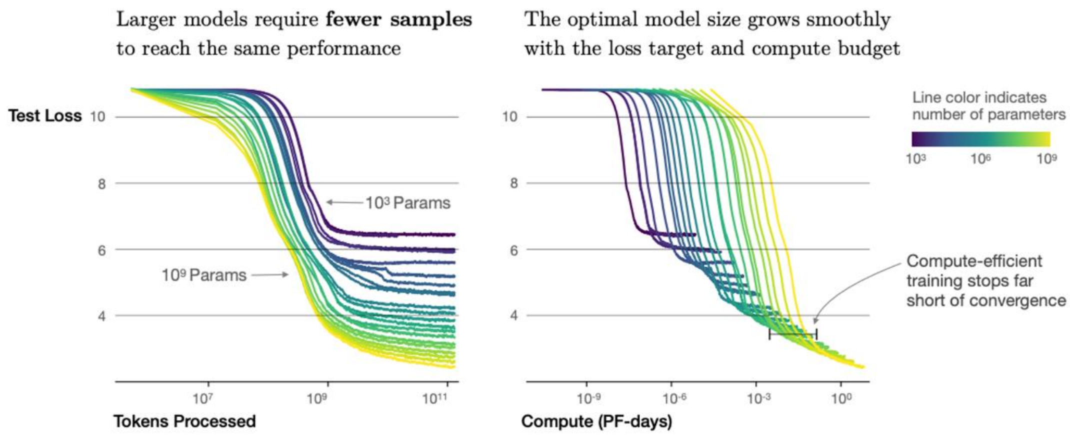
The paper differentiates between **embedding and non-embedding parameters** because their size correlates differently with model performance. When including embedding parameters, performance appears to depend strongly on the number of layers in addition to the number of parameters. When excluding embedding parameters, the performance of models with different depths converges to a single trend.

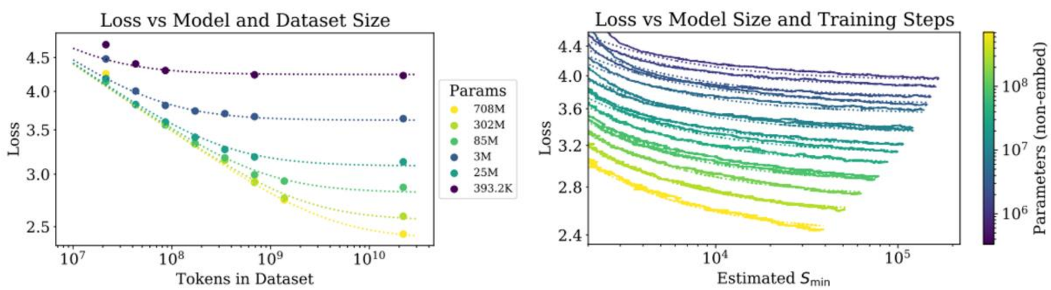


- The **universality of overfitting:** Performance improves predictably as long as we scale up *N* and *D* in tandem but leads to diminishing returns if either *N* or *D* is held fixed while the other increases.

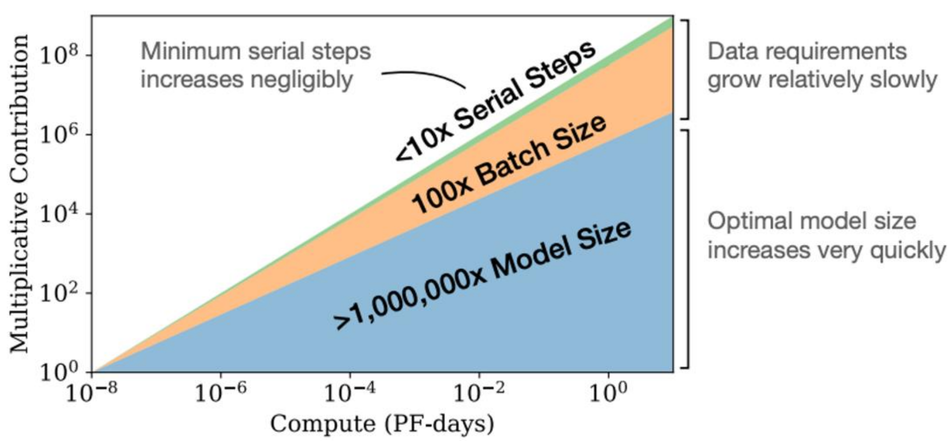


- The **universality of training**: Training curves follow predictable power-laws whose parameters are roughly independent of the model size. Extrapolating the early part of a training curve can roughly predict the loss that would be achieved if trained for much longer.
- **Sample efficiency**: Large models are more sample-efficient than small models, reaching the same level of performance with fewer optimization steps and data points.





- **Convergence is inefficient**: When working within a fixed compute budget *C* but without any other restrictions on the model size *N* or available data *D*, we attain optimal performance by training very large models and stopping significantly short of convergence.



These results show that language modeling performance improves smoothly and predictably as we appropriately scale up model size, data, and compute. Conversely, we find very weak dependence on many architectural and optimization hyperparameters. Larger language models are expected to perform better and be more sample-efficient than current models.

Considerations

When training large language models, it's possible to use the relations between *N*, *D*, and *L* to derive the compute scaling, magnitude of overfitting, early stopping step, and data requirements.

The derived scaling relations can be used as a predictive framework. One might interpret these relations as analogs of the ideal gas law, which relates the macroscopic properties of a gas in a universal way, independent of most of the details of its microscopic constituents.

It would be interesting to investigate whether these scaling relations hold in other generative modeling tasks with a maximum likelihood loss and perhaps in other settings and domains (such as images, audio, and video models) as well.

Chinchilla Scaling Laws for Compute-Optimal Training of LLMs

In 2022, Google DeepMind published the paper "Training Compute-Optimal Large Language Models" that further explored the scaling laws of LLMs. The researchers conducted extensive experiments to understand the relationship between model size, the number of training tokens, and the compute budget.

The key finding of this study was that current LLMs, such as GPT-3 (175B), Gopher (280B), and Megatron (530B), are significantly undertrained. While these models have increased the number of parameters, the training data remained constant.

The authors proposed that the number of training tokens and model size must be scaled equally for compute-optimal training. They trained approximately 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens. This extensive experimentation led to the creation of a new LLM, Chinchilla, which outperformed its larger counterparts.

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

With 70B parameters and four times more training data, Chinchilla was trained using the same compute budget as the 280B Gopher. The results showed that smaller models could deliver better performance if trained on more data. These smaller models are easier to fine-tune and have less latency at inference. Moreover, they do not need to be trained to their lowest possible loss to be compute optimal.

The researchers explored three different approaches to answer the question: "Given a fixed FLOPs budget, how should one trade-off model size and the number of training tokens?" They assumed a power-law relationship between compute and model size.

1. The first approach involved fixing model sizes and varying the number of training tokens.
2. The second approach, called IsoFLOP profiles, varied the model size for a fixed set of different training FLOP counts.
3. The third approach combined the final loss of the above two approaches as a parametric function of model parameters and the number of tokens.

All three approaches suggested that as the compute budget increases, the model size and the training data amount should be of approximately equal proportions. The first and second approaches yielded similar predictions for optimal model sizes, while the third suggested that smaller models would be optimal for larger compute budgets.

Conclusion

This lesson has explored the relationship between language model performance and parameters such as model size, dataset size, and compute budget.

We've learned that performance scales are a power law with these variables and larger models tend to be more sample-efficient. We also explored the Chinchilla Scaling Laws, which suggest that the number of training tokens and model size should be scaled equally for compute-optimal training. This has led to the creation of smaller models, like Chinchilla, that outperform larger counterparts when trained on more data.

These findings provide a predictive framework for training large language models and may have implications for other generative modeling tasks and domains.

ReAct framework and ChatGPT plugins

Introduction

Large Language Models have demonstrated great utility in diverse tasks, from coding assistance and content summarization to answering everyday questions. As our understanding of their strengths and limitations grows, numerous innovative methods for improvement and expanding their range of tasks have emerged in recent months.

This lesson will delve into some of these advancements. We will learn about the ReAct framework, a **prompt-based paradigm** designed to synergize **reasoning** and **acting** in language models for **general task solving**.

Additionally, this module will cover the latest upgrades to ChatGPT, including plugins integration. We will also explore new enhancements available through the OpenAI API, such as function calling. This feature enables LLMs to produce structured outputs, further augmenting their reliability and utility in many applications.

Overview of ReAct

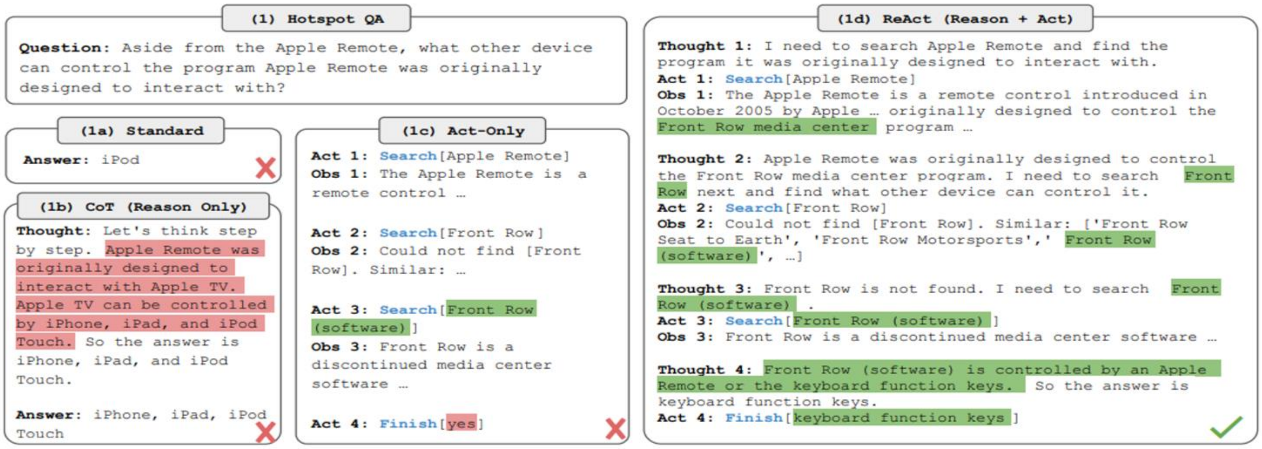
This framework aims to enhance the utility of language models by using them to create **autonomous agent systems**, which are systems that can operate and make decisions independently. A way to accomplish that is to make these models reason about an input question and context, create an action plan, and execute it.

In the **ReAct framework**, language models are prompted to generate **verbal reasoning traces,** which are detailed records of the model's thought process, and **actions** interleaved when accomplishing a task. The process is done iteratively until the answer to a question is found.

The verbal reasoning step in ReAct allows the model to dynamically create, maintain, and adjust high-level plans for acting, which refers to the execution of specific tasks or actions. When acting, the model can also interact with external environments (like Wikipedia) to incorporate additional information into reasoning.

As a side note, allowing these models to access sources of information like Wikipedia can also reduce the number of hallucinations and biases.

To better understand the framework, let’s look at the example below from the paper.



From "[ReAct: Synergizing Reasoning and Acting in Language Models]" paper

The figure above provides a comparative analysis of different methods a language model is utilized to accomplish a specific task: identifying a device that can control the same program the Apple Remote was first designed to interact with.

- (1a) The model is asked to answer the question directly.
- (1b) Uses **Chain-of-Thought** prompting, which asks the LLM to reason about the question before answering.
- (1c) An act-only process where the LLM is not prompted to reason about the question.
- (1d) Using the ReAct framework, the LLM is prompted to reason about the question and perform a specific action to find the answer.

The authors of ReAct note that the framework recipe accomplishes various tasks. The main steps include:

1. **Thought Step**: The LLM is prompted to think critically about the task. Given the question, it evaluates which actions might lead to finding the answer.
2. **Action Steps**: In this phase, the LLM interacts with an external environment. It can utilize external APIs to acquire necessary information if needed.
3. **Observation Step**: After taking action, the LLM receives a result from the external environment. These observations are crucial for the LLM to determine the effectiveness of the action and plan the next steps.
4. **Next Thought Step**: Equipped with the information from the action and observation, the LLM reevaluates the situation. This evaluation allows the model to consider and decide on the subsequent action.

This sequential process continues until the LLM successfully finds the answer.

ReAct framework in code

Code implementations of the ReAct framework are available for those interested in creating autonomous agents. For a practical demonstration, consider looking at the **author's implementation**. This link directs you to a notebook showcasing an example of utilizing `text-davinci-002` to create an agent that answers questions using Wikipedia as a source of information. There is also a **LangChain** implementation of ReAct, enabling you to create a capable agent in less time as they have many available agent tools.

OpenAI Function calling

OpenAI has recently introduced a [function calling](#) feature for their language models through their API.

In an API call, you can describe functions to `gpt-3.5-turbo-0613` and `gpt-4-0613`, and have the model output a JSON object containing arguments to call those functions. Be aware that the Chat completions API **does not call the function**; instead, the model generates a JSON object that you can use to **call the function in your code**.

To include this feature, OpenAI fine-tuned the models `gpt-3.5-turbo-0613` and `gpt-4-0613` to detect when a function should be called (depending on the user input) and to respond with a JSON object that adheres to the function signature. It's not disclosed how they implement this, but they may be using a form of prompt engineering similar to ReAct for this.

With this feature, you can more easily create:

- Chatbots that answer questions by calling external tools, such as sending an email.
- Convert natural language into API calls or database queries so the model can answer questions such as "Who are my top ten customers this month?"
- Extract structured data from text, such as extracting the names of all the locations mentioned in a Wikipedia article.

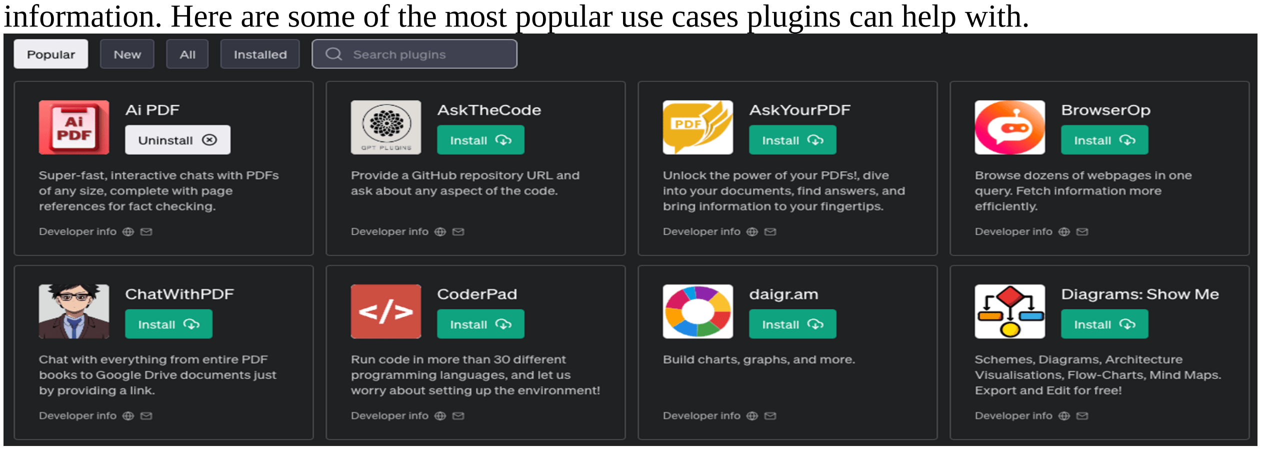
Check out this example in the OpenAI documentation to learn how to set up function calling.

If your application makes use of LangChain, there is also a way to use function calling; take a look at the documentation here.

ChatGPT Plugins

If you are subscribed to ChatGPT Plus, you can easily augment LLMs with tools for personal or professional use without using the Open AI API. They made available the use of third-party Plugins through their chat interface. It's not disclosed how OpenAI implement this, but they may be using a form of prompt engineering similar to ReAct to abilitate the plugins.

These plugins or tools can give their language models access to more recent, personal, or specific information. Here are some of the most popular use cases plugins can help with.



Plugins available to ChatGPT Plus subscribers

With these third-party plugins, you can directly upload your documents, such as PDFs, and ask questions about the information in those documents. You can provide a link to a GitHub repository and ask questions or let the language model explain the code to you. There are also plugins to create diagrams, flow charts, or graphs.

Conclusion

In this module, we explore various ways to enhance the capabilities of current language models. These new methods also reduce the risks of hallucinations. We learn about the ReAct framework, a tool that empowers language models to act independently. We discuss the diverse features available via OpenAI services, including function calling and third-party plugins. Function calling assists developers by allowing the use of custom functions during user interaction with the model. Concurrently, plugins available through the chat interface enable users to utilize enhanced OpenAI language models without coding.

Expanding the Context Window

Introduction

In this lesson, we will discuss context windows in language models, their importance, and the limitations of the original Transformer architecture in handling large context lengths.

We explore various optimization techniques that have been developed to expand the context window, including ALiBi Positional Encoding, Sparse Attention, FlashAttention, Multi-Query Attention, and the use of large RAM GPUs.

We also introduce the latest advancements in this field, such as FlashAttention-2 and LongNet, which aim to push the context window to an unprecedented scale.

The Importance of The Context Length

The context window refers to the **number of input tokens** the model can process simultaneously. In current models like GPT-4, this context window is around 32K tokens. To put this into perspective, this roughly translates to the size of 50 pages. However, recent advancements have pushed this limit to an impressive 100K tokens (check Claude by Anthropic), equivalent to 156 pages.

The context length of an LLM is a critical factor for several reasons. Firstly, it allows the model to process larger amounts of data at once, providing a more comprehensive understanding of the context. This is particularly useful when you want to feed a large amount of custom data into an LLM and ask questions about this specific data.

For instance, you might want to input a large document related to a specific company or problem and ask the model questions about this document. With a larger context window, the LLM can scan and retain more of this custom information, leading to more accurate and personalized responses.

Limitations of the Original Transformer Architecture

The original Transformer architecture, however, has some limitations when it comes to handling large context lengths. The main issue lies in the computational complexity of the Transformer architecture. Specifically, the attention layer computations in the Transformer architecture have a quadratic time and space complexity with respect to the number of input tokens n. This means that as the **context length increases, the computational resources required for training and inference increase exponentially**.

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

This complexity arises from two main operations in the attention layer: **linear projections** to get **Query, Key, and Value matrices** (complexity **~ $O(nd^2)$**) and **multiplications** of these matrices (complexity ~ **$O(n^2d)$).** As the context length or embedding size increases, the computational complexity grows quadratically, making it increasingly challenging to process larger context lengths.

Optimization Techniques to Expand the Context Window

Despite these challenges, researchers have developed several optimization techniques to speed up the Transformer and increase the context length to 100K tokens. Let's explore some of these techniques:

1. **ALiBi Positional Encoding**: The original Transformer uses Positional Sinusoidal Encoding, which lacks the ability to extrapolate to larger context lengths. ALiBi, or Attention with Linear Biases, is a positional encoding technique that can be used to train the model on a small context and then fine-tune it on a larger one.
2. **Sparse Attention**: This technique reduces the number of computations by considering only some tokens when calculating the attention scores. This makes the computation linear with respect to n, significantly reducing the computational complexity.
3. **FlashAttention**: This is an efficient implementation of the attention layer for GPU. It optimizes the memory utilization of the GPU by splitting the input matrices into blocks and computing the attention output with respect to these blocks.
4. **Multi-Query Attention (MQA)**: MQA optimizes the memory consumption of the key/value decoder cache by sharing weights across all attention heads when linearly projecting Key and Value matrices.
5. **Large RAM GPUs**: You need a lot of RAM in the GPU to fit a large context. Therefore, models with larger context windows are often trained on GPUs with large RAM, such as 80GB A100 GPUs.

FlashAttention-2

Building on the success of FlashAttention, researchers have recently developed FlashAttention-2, a more efficient version of the algorithm that further optimizes the attention layer's speed and memory usage. This new version has been completely rewritten from scratch, leveraging the new primitives from Nvidia. The result is a version that is about 2x faster than its predecessor, reaching up to 230 TFLOPs/s on A100 GPUs.

FlashAttention-2 introduces several improvements over the original FlashAttention.

- Firstly, it reduces the number of non-matmul FLOPs, which are 16x more expensive than matmul FLOPs, by tweaking the algorithm to spend more time on matmul FLOPs.
- Secondly, it optimizes parallelism by parallelizing over batch size, number of heads, and the sequence length dimension. This results in significant speedup, especially for long sequences.
- Lastly, it improves work partitioning within each thread block to reduce the amount of synchronization and communication between different warps, resulting in fewer shared memory reads/writes.
- In addition to these improvements, FlashAttention-2 also introduces new features, such as support for head dimensions up to 256 and multi-query attention (MQA), further expanding the context window.

With these advancements, FlashAttention-2 is a step forward in expanding the context window (without overcoming the fundamental limitations of the original Transformer architecture).

LongNet: A Leap Towards Billion-Token Context Window

Building on the advancements in Transformer optimization, a recent innovation comes from the paper "LONGNET: Scaling Transformers to 1,000,000,000 Tokens". This paper introduces a novel approach to handling the computational complexity of the Transformer architecture, pushing the context window potentially to an unprecedented 1 billion tokens.

The core innovation in LongNet is the introduction of "**dilated attention**." This novel attention mechanism expands the attentive field exponentially as the distance between tokens grows, thereby decreasing attention allocation exponentially as the distance increases. This design principle helps to balance the limited attention resources with the necessity to access every token in the sequence.

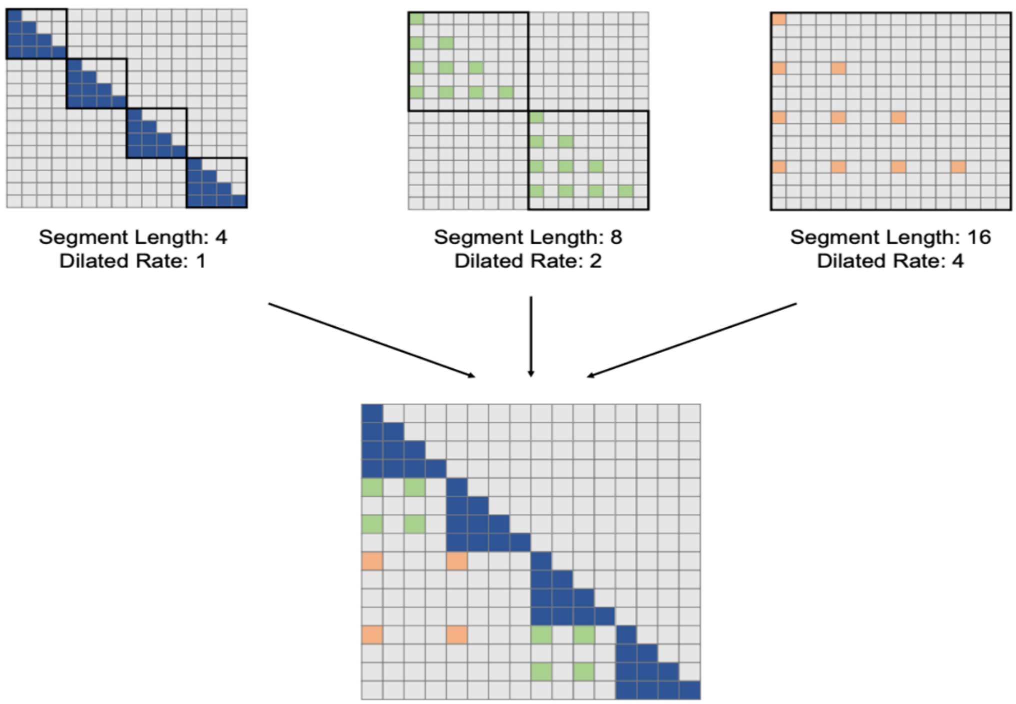


Image from the paper "LONGNET: Scaling Transformers to 1,000,000,000 Tokens"

Building blocks of dilated attention used in LONGNET. It consists of a series of attention patterns for modeling short- and long-range dependency. The number of attention patterns can be extended according to the sequence length.

The dilated attention mechanism in LongNet achieves a **linear computational complexity**, a significant improvement over the quadratic complexity of the standard Transformer.

Method	Computation Complexity
Recurrent	$\mathcal{O}(Nd^2)$
Vanilla Attention	$\mathcal{O}(N^2d)$
Sparse Attention	$\mathcal{O}(N\sqrt{N}d)$
Dilated Attention (This Work)	$\mathcal{O}(Nd)$

Image from the paper "LONGNET: Scaling Transformers to 1,000,000,000 Tokens".

Comparison of computation complexity among different methods. N is the sequence length, and d is the hidden dimension.

Conclusion

In this lesson, we examined the limitations of the original Transformer architecture in handling large context lengths, primarily due to its quadratic computational complexity. We then explored various optimization techniques developed to overcome these limitations, including **ALiBi Positional Encoding, Sparse Attention, FlashAttention, Multi-Query Attention**, and the use of **large RAM GPUs.**

We also discussed the latest advancements in this field, such as **FlashAttention-2**, which further optimizes the speed and memory usage of the attention layer, and **LongNet**, a novel approach that introduces "**dilated attention**" to potentially expand the context window to an unprecedented 1 billion tokens.

These advancements are critical in pushing the boundaries of language models, enabling them to process larger amounts of data at once and providing a more comprehensive understanding of the context, leading to more accurate and personalized responses.

Training on Generated Data and Model Collapse

Introduction

In this lesson, we will examine the phenomenon of model collapse, its stages, its causes, and its implications on the future of Large Language Models. We also draw parallels with related concepts in machine learning, such as **catastrophic forgetting and data poisoning**. Finally, we contemplate the value of human-generated content in the era of dominant LLMs and the potential risk of widespread model collapse.

Understanding Model Collapse

Model collapse, defined in the paper "[The Curse of Recursion: Training on Generated Data Makes Models Forget](),” is a degenerative process affecting generations of learned generative models. It occurs when the data generated by a model ends up contaminating the training set of subsequent models. As a result, these models start to misinterpret reality, reinforcing their own beliefs instead of learning from real data.

Here’s an image exemplifying model collapse.

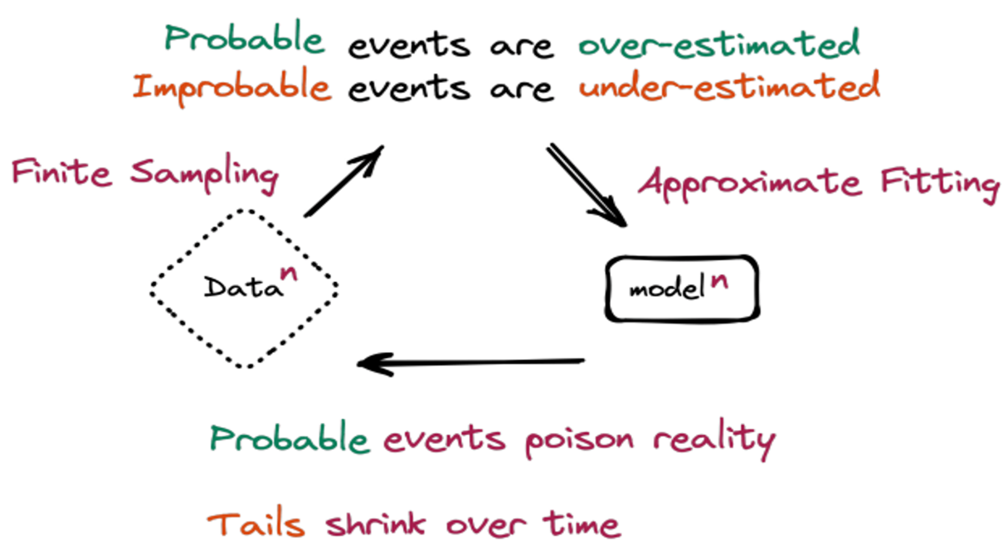


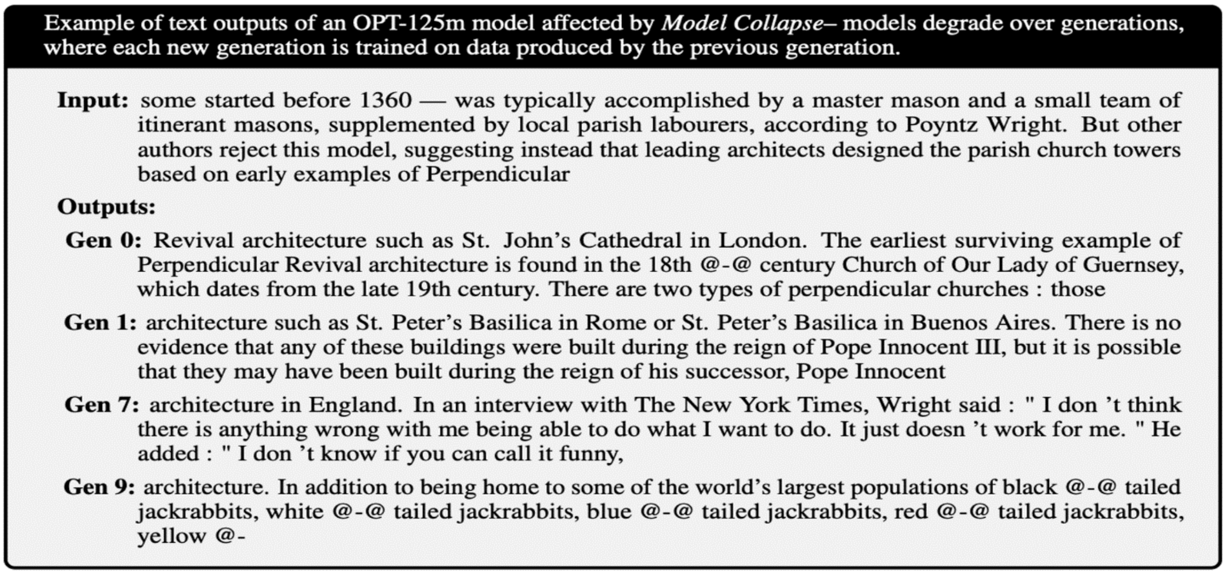
Image from the paper “The Curse of Recursion: Training on Generated Data Makes Models Forget.”

Model Collapse refers to a **degenerative learning process** where models start forgetting improbable events over time as the model becomes poisoned with its projection of reality.

There are two distinct stages of model collapse: early and late.

- In the **early stage**, the model begins to lose information about the tails of the distribution.
- As the process progresses to the **late stage**, the model starts to entangle different modes of the original distributions, eventually converging to a distribution that bears little resemblance to the original one, often with very small variance.

Here's an example of text outputs from sequential generations of 125M parameters LLMs where each generation is trained on data produced by the previous generation.



Example of text outputs of an OPT-125m model affected by *Model Collapse*– models degrade over generations, where each new generation is trained on data produced by the previous generation.

Input: some started before 1360 — was typically accomplished by a master mason and a small team of itinerant masons, supplemented by local parish labourers, according to Poyntz Wright. But other authors reject this model, suggesting instead that leading architects designed the parish church towers based on early examples of Perpendicular

Outputs:

Gen 0: Revival architecture such as St. John's Cathedral in London. The earliest surviving example of Perpendicular Revival architecture is found in the 18th @-@ century Church of Our Lady of Guernsey, which dates from the late 19th century. There are two types of perpendicular churches : those

Gen 1: architecture such as St. Peter's Basilica in Rome or St. Peter's Basilica in Buenos Aires. There is no evidence that any of these buildings were built during the reign of Pope Innocent III, but it is possible that they may have been built during the reign of his successor, Pope Innocent

Gen 7: architecture in England. In an interview with The New York Times, Wright said : " I don 't think there is anything wrong with me being able to do what I want to do. It just doesn 't work for me. " He added : " I don 't know if you can call it funny,

Gen 9: architecture. In addition to being home to some of the world's largest populations of black @-@ tailed jackrabbits, white @-@ tailed jackrabbits, blue @-@ tailed jackrabbits, red @-@ tailed jackrabbits, yellow @-

Image from the paper "The Curse of Recursion: Training on Generated Data Makes Models Forget."

Related Work on Model Collapse

Model collapse shares similarities with two concepts in machine learning literature: catastrophic forgetting and data poisoning.

- Catastrophic forgetting, a challenge in continual learning, refers to the model's tendency to forget previous samples when learning new information. This is particularly relevant in task-free continual learning, where data distributions gradually change without the notion of separate tasks. However, in the context of model collapse, the changed data distributions arise from the model itself as a result of training in the previous iteration.
- On the other hand, data poisoning involves the insertion of malicious data during training to degrade the model's performance. This concept becomes increasingly relevant with the rise of contrastive learning and LLMs trained on untrustworthy web sources.

Yet, neither catastrophic forgetting nor data poisoning fully explain the phenomenon of model collapse, as they don't account for the self-reinforcing distortions of reality seen in model collapse. However, understanding these related concepts can provide additional insights into model collapse mechanisms and potential mitigation strategies.

Causes of Model Collapse

Model collapse primarily results from two types of errors: **statistical approximation error** and **functional approximation error**.

- The statistical approximation error is the primary cause. It arises due to the finite number of samples used in training. Despite using a large number of points, significant errors can still occur. This is because there's always a non-zero probability that information can get lost at every step of re-sampling.
- The functional approximation error is a secondary cause. It stems from the limitations of our function approximators. Even though neural networks are theoretically capable of approximating any function, in practice, they can introduce non-zero likelihood outside the support of the original distribution, leading to errors.

The Future of the Web with Dominant LLMs

As LLMs become more prevalent in the online text and image ecosystem, they will inevitably train on data produced by their predecessors. This could lead to a cycle where each model generation learns more from previous models' output and less from original human-generated content. The result is a risk of widespread model collapse, with models progressively losing touch with the true underlying data distribution.

The model collapse has far-reaching implications. As models start to misinterpret reality, generated content quality could degrade over time. This could profoundly affect various LLMs' applications, from content creation to decision-making systems.

The Value of Human-Generated Content

In the face of model collapse, preserving and accessing data collected from genuine human interactions becomes increasingly valuable. Real human-produced data provides access to the original data distribution, which is crucial in learning where the tails of the underlying distribution matter. As LLMs increasingly generate online content, data from human interactions with these models will become an increasingly valuable resource for training future models.

Conclusion

In this lesson, we've explored the phenomenon of model collapse, a degenerative process that can affect generative models when they are trained on data produced by other models.

We've examined the stages of model collapse, from the early loss of information about the tails of distribution to the late-stage entanglement of different modes. We've drawn parallels with related concepts in machine learning: catastrophic forgetting and data poisoning. We also dissected the leading causes of model collapse, namely statistical and functional approximation errors.

As Large Language Models become more dominant in the digital landscape, the risk of widespread model collapse increases, potentially leading to a degradation in generated content quality. In this context, we've underscored the importance of preserving and accessing human-generated content, which provides a crucial link to the original data distribution and serves as a valuable resource for training future models.

As we continue to harness the power of LLMs, understanding and mitigating model collapse will be essential in ensuring the quality and reliability of their outputs.

Next Challenges in LLM Research

Introduction

In this lesson, we will view the next challenges in large language model research, covering various facets such as model performance, data and training, language and tokenization, hardware and infrastructure, usability and application, and learning and preferences.

We will explore pressing issues such as mitigating hallucinations, optimizing context, managing massive datasets, improving tokenization, and developing alternatives to GPUs. We will also discuss the need to make agents usable, detect LLM-generated text, and improve learning from human preference.

Model Performance and Efficiency

- **Mitigating and Measuring Hallucinations**: One of the significant challenges in LLM research is hallucinations. This phenomenon occurs when an AI model generates information that isn't based on the input data, essentially making things up. While this can be beneficial for creative applications, it is generally considered a drawback for most use cases. The challenge lies in reducing and developing metrics to measure these hallucinations accurately.
- **Optimizing Context Length and Construction**: Context plays a crucial role in the performance of LLMs. The challenge here is to optimize the context length and how it is constructed. This is particularly important for applications like Retrieval Augmented Generation (RAG), where the model's response quality depends on the amount and efficiency of the context it can use.
- **Making LLMs Faster and Cheaper**: With the advent of models like GPT-3.5, concerns about latency and cost have become more prominent. The challenge lies in developing models that offer similar performance but with a smaller memory footprint and lower costs. Faster inference is especially important for real-time applications like online customer service assistants.
- **Designing New Model Architectures**: Transformer architecture has been dominant in the field since 2017. However, the need for a new model architecture that can outperform the Transformer is becoming increasingly apparent. The challenge is to develop an architecture that performs well on current hardware and scales to meet modern requirements.
- **Addressing High Inference Latency**: LLMs often exhibit high inference latencies due to low parallelizability and large memory footprints. The task at hand is to develop models and techniques that can reduce this latency, making LLMs more efficient and practical for real-time applications.
- **Overcoming Tasks Not Solvable By Scale**: The rapid advancements in LLM capabilities have led to astonishing improvements in performance. However, some tasks seem resistant to further scaling of data or model sizes. The existence of such tasks is speculative, but their potential presence poses a significant challenge. The research community needs to identify these tasks and devise strategies to overcome them, pushing the boundaries of what LLMs can achieve. Read about the Inverse Scaling Prize competition to know more about this.

Data and Training

- **Incorporating Other Data Modalities**: The ability to incorporate other data modalities into LLMs is another significant research direction. Multimodality, the ability to understand and process different types of data, can enhance the performance of LLMs and extend their applicability to various industries.
- **Understanding and Managing Huge Datasets**: The sheer size of modern pre-training datasets makes it nearly impossible for individuals to read or conduct quality assessments on all the documents. This lack of clarity about the data on which the model has been trained poses a significant challenge. Researchers need to devise strategies to comprehend these vast datasets better and ensure the quality of the data used for training.
- **Reducing High Pre-Training Costs**: Training a single LLM can require substantial computational resources, translating into high costs and significant energy consumption. The challenge here is to find ways to reduce these pre-training costs without compromising the performance of the model. This could involve optimizing the training process or developing more efficient model architectures.

Language and Tokenization

- **Building LLMs for Non-English Languages**: There is a pressing need to develop LLMs for non-English languages. This complex challenge involves dealing with low-resource languages and ensuring that the models are practical and efficient.
- **Overcoming Tokenizer-Reliance**: Tokenization, the process of breaking down text into smaller units, is crucial for feeding data into the model. However, this necessity comes with drawbacks, such as computational overhead, language dependence, handling of novel words, fixed vocabulary size, information loss, and low human interpretability. The challenge lies in developing more effective tokenization methods or alternatives that can mitigate these issues.
- **Improving Tokenization for Multilingual Settings**: Tokenization schemes that work well in a multilingual setting, particularly with non-space-separated languages such as Chinese or Japanese, remain challenging. The challenge is to improve these schemes to ensure fair and efficient tokenization across all languages.

Hardware and Infrastructure

- **Developing Alternatives to GPUs**: GPUs have been the primary hardware for deep learning for nearly a decade. However, there is a growing need for alternatives that can offer better performance or efficiency. This includes exploring technologies like quantum computing and photonic chips. There's also currently a problem of availability of GPUs in the global market, therefore having alternatives would make this problem more manageable.

Usability and Application

- **Making Agents Usable**: Agents are LLMs that can perform actions like browsing the internet or sending emails. The challenge here is to make these agents reliable and performant enough to be trusted with these tasks. Examples of agents frameworks are LangChain and LlamaIndex.
- **Detecting LLM-generated Text**: As LLMs become more sophisticated, distinguishing between human-written and LLM-generated text becomes increasingly challenging. This detection is crucial for various reasons, such as preventing the spread of misinformation, plagiarism, impersonation, automated scams, and the inclusion of inferior generated text

in future models' training data. The challenge lies in developing robust detection mechanisms that can keep up with the improving fluency of LLMs.

Learning and Preferences

- **Improving Learning from Human Preference**: Reinforcement Learning from Human Preference (RLHF) is a promising approach but has its challenges. These include defining and mathematically representing human preferences and dealing with the diversity of human preferences.

Conclusion

In this lesson, we explored the challenges in large language models research.
We've examined the need for improved model performance and efficiency, including mitigating hallucinations, optimizing context, and designing new model architectures.
We've also discussed the complexities of managing vast datasets and the importance of incorporating other data modalities.
We highlighted the necessity for better tokenization methods, especially for non-English and non-space-separated languages.
We have also underscored the urgency of developing alternatives to GPUs and the need to make LLM agents more reliable.
Lastly, we've touched upon the challenge of detecting LLM-generated text and the intricacies of learning from human preference.
Each of these challenges presents an exciting opportunity for researchers to push the boundaries of what LLMs can achieve, making them more efficient, inclusive, and beneficial for a wide array of applications.