



AMERICAN
UNIVERSITY
OF BEIRUT

Fall 2025

EECE 503P/798S: Agentic Systems

C2 - Large Language Models



Reflection on course feedback

Project and Assignment Challenges

- Students find coming up with innovative project ideas difficult.
 - “Help us shape the project”
- Some feel the first assignment is too theoretical and not well aligned with in-class explanations.
 - “Assignment doesn't match what is explained at all in class”

Pace and Complexity of Material

- Intro and Outro to be given by me.
- Several students mentioned the pace feels fast and some concepts are not intuitive, requiring multiple reviews.
- Those without a strong AI background struggled more with grasping theory.

Uncertainty Around Structure & Expectations

- Comments highlighted unclear exam structure and confusion about how to start projects early when Agentic Systems concepts are still being introduced.
 - “Being unaware of the exam structure”

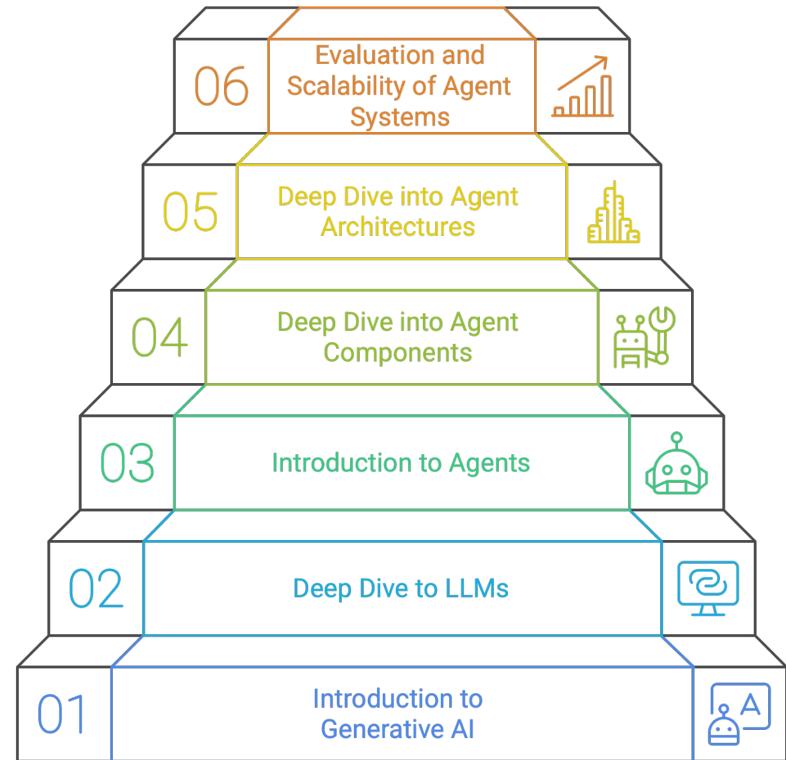


Lesson Objectives

- Define **LLMs** and trace their **evolution** from earlier generative AI.
- Explain what makes a model “**large**” (parameters, data, compute).
- Explore **emergent behaviors** beyond text prediction.
- Learn **key prompting techniques** (zero-shot, few-shot, CoT, multi-step, style).
- Compare **LLM architectures** (encoder-only, decoder-only, encoder–decoder, MoE, multimodal).
- Review **recent advancements** (context windows, scaling laws, reasoning).
- Introduce **fine-tuning** and **evaluation** (instruction tuning, RLHF, DPO, benchmarks, safety).
- Distinguish **open- vs. closed-source LLMs** and highlight Hugging Face.



Course Timeline



We are here



AMERICAN
UNIVERSITY
OF BEIRUT

Introduction



Evolution of Generative AI Models

Early Foundations

Markov chains in early 20th century set groundwork for probabilistic models.

1

VAEs/GANs (2014)

Introduced powerful generative techniques for images and beyond.

2

Deep Learning Rise

Late 2000s breakthroughs enabled complex pattern recognition.

3

LLMs (2022)

OpenAI's release sparked rapid innovation and widespread adoption.

4

Transformer (2017)

Revolutionized natural language understanding and generation.

5



AMERICAN
UNIVERSITY
OF BEIRUT

From Transformers to LLMs



What are LLMs?

- Neural networks trained on **massive text corpora** to model language distribution.
- Capable of **tasks** like text generation, translation, summarization, question answering
- **Are all LLMs transformer-based?**
No. We will get to see many variations in the upcoming sections

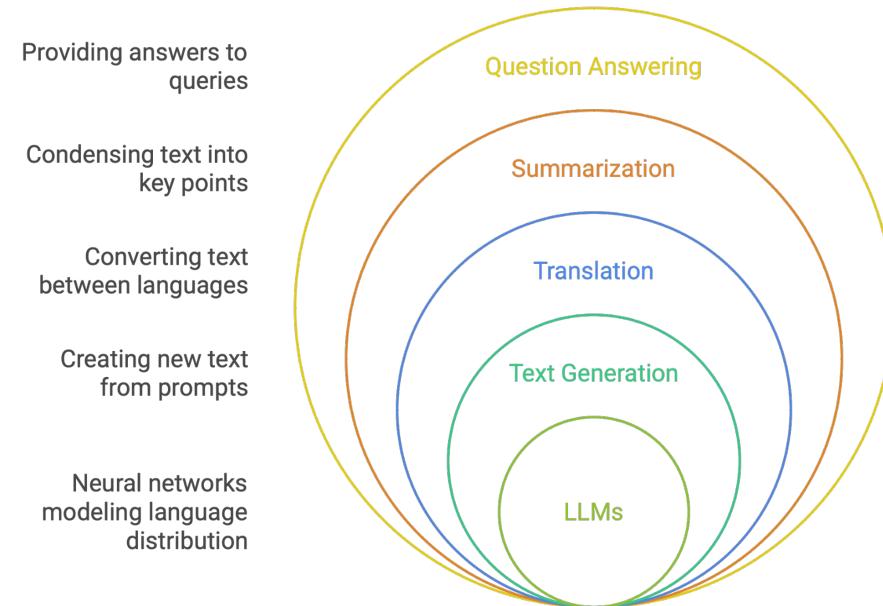
Providing answers to queries

Condensing text into key points

Converting text between languages

Creating new text from prompts

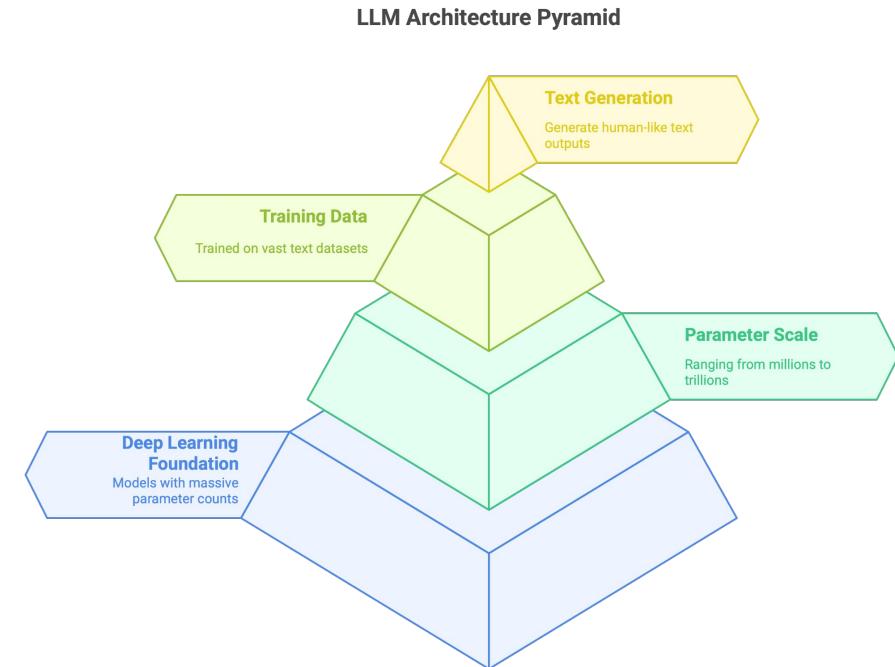
Neural networks modeling language distribution





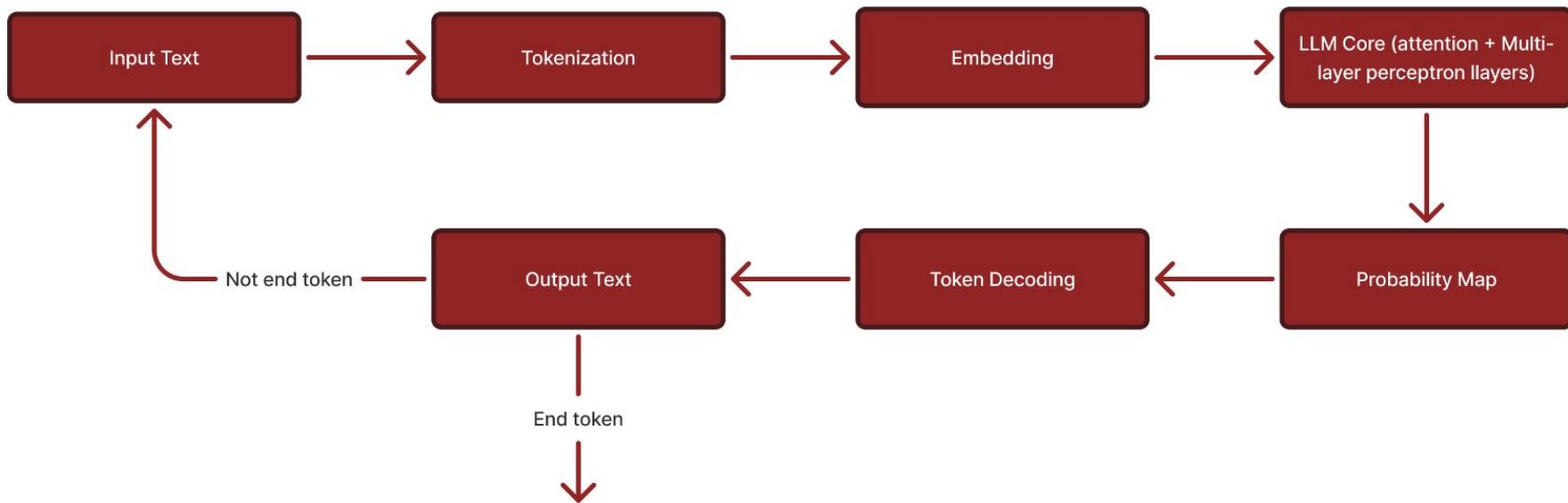
What makes a language model “large”?

- The original transformer architecture, consisting or an **encoder** and a **decoder** block, is a **language model**. It is not a large language model.
- However, if you stack up **hundreds** of encoder blocks and hundreds of decoder blocks until you get **millions- even billions-** of training parameters, the language model is considered large.





LLM Input and output





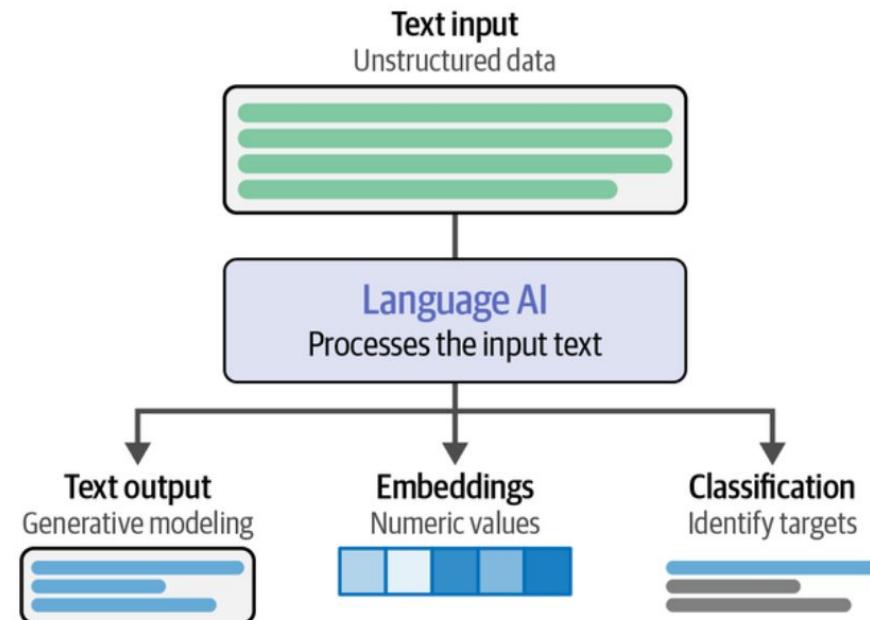
AMERICAN
UNIVERSITY
OF BEIRUT

Why were LLMs even created?



Language Representation

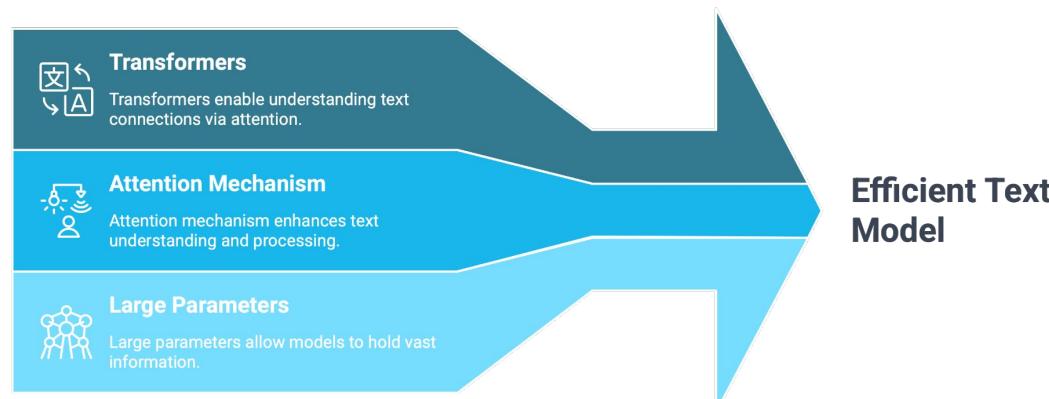
- Language is **challenging** for computers due to its **unstructured nature**.
- Text **loses meaning** when reduced to zeros and ones (individual characters).
- Language AI has historically focused on **structuring language** for easier computer processing.





From Transformers to Large Models

- To have one text model that is extremely efficient in answering any type of question, creative writing, coding, and multi-language generation, we would need to hold a **lot of information** within its weights.
- To have a model that is able to contain a lot of information, we need a **large number of parameters**





Emergent Behaviours of LLMs

The original purpose of large language models was to perform traditional text tasks with **high accuracy**

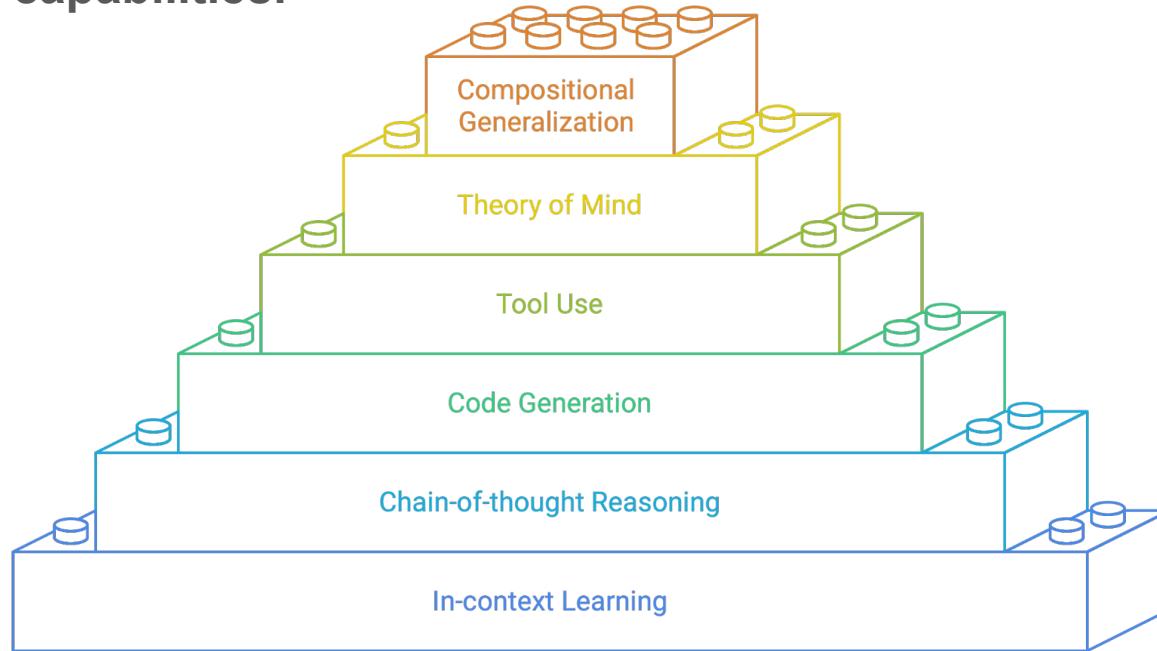
Original Purpose of Large Language Models

Characteristic	Description
Predictive Text Modeling	Trained to predict next word
Language Understanding	Capture patterns based on context
Initial Applications	Autocomplete, translation, classification, summarization



Emergent Behaviours of LLMs

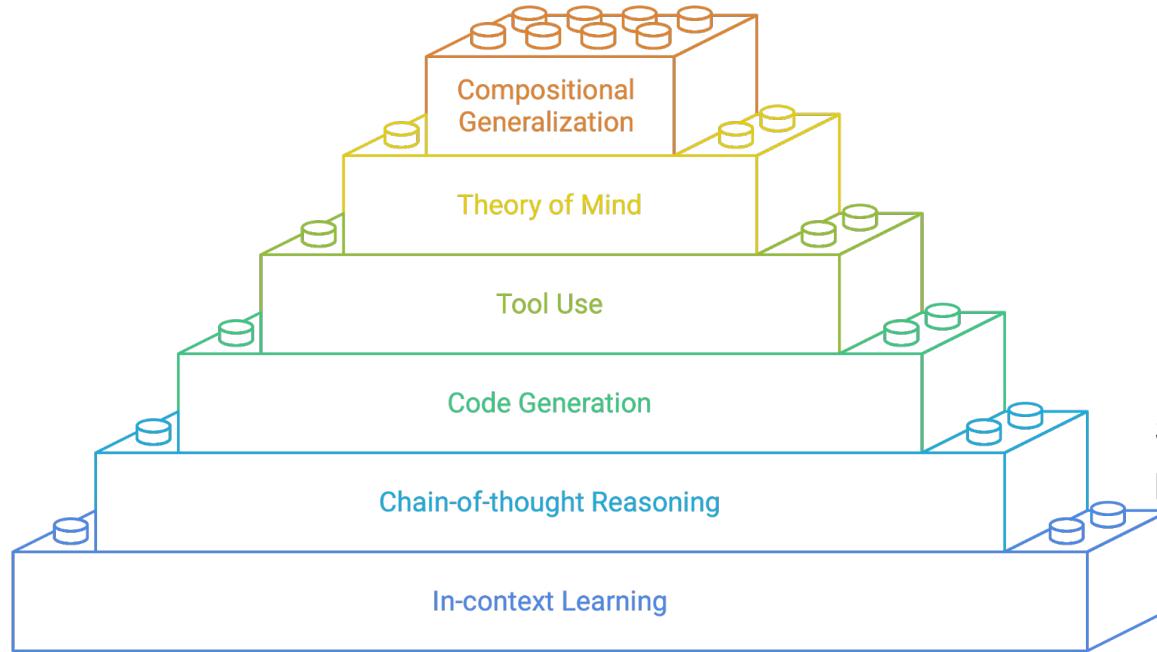
Despite being trained purely to predict text, LLMs unexpectedly **gained powerful capabilities**.



Solving new tasks without gradient updates, just from prompts.



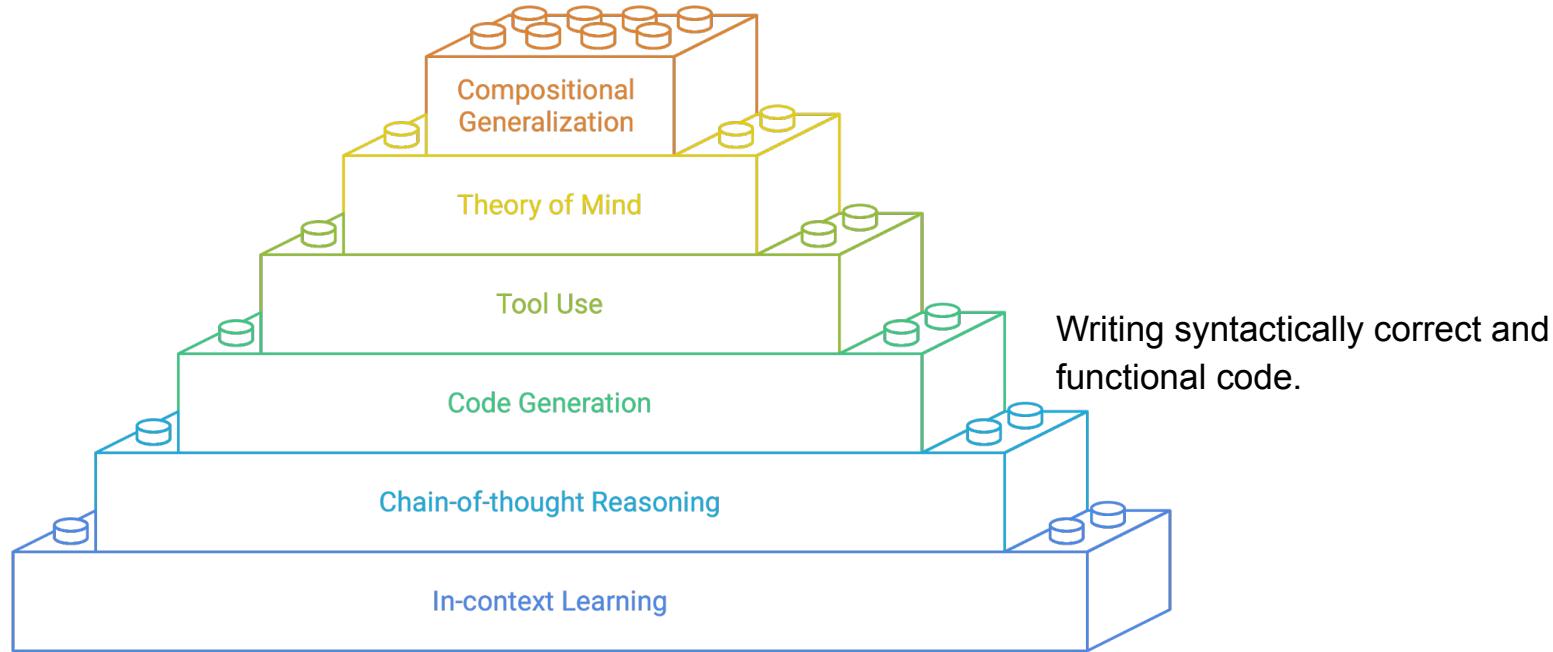
Emergent Behaviours of LLMs



Step-by-step logic emerges when prompted correctly.

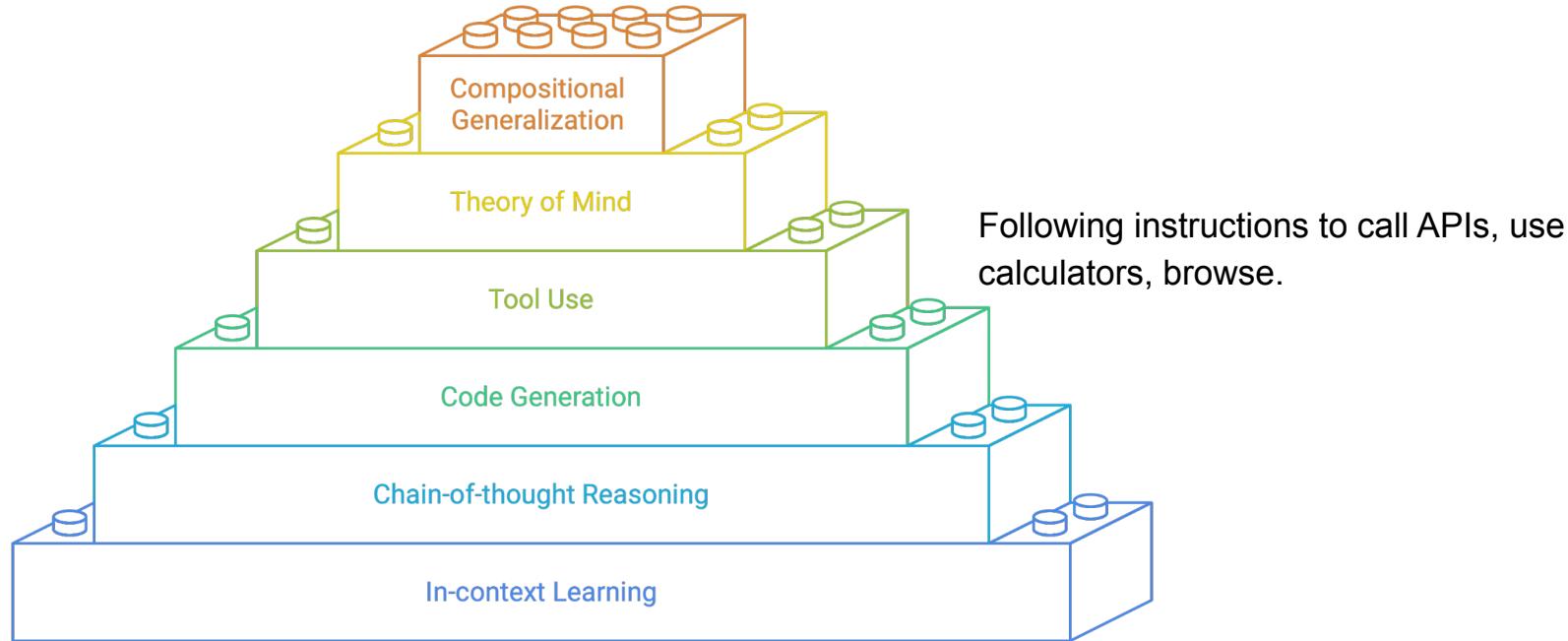


Emergent Behaviours of LLMs



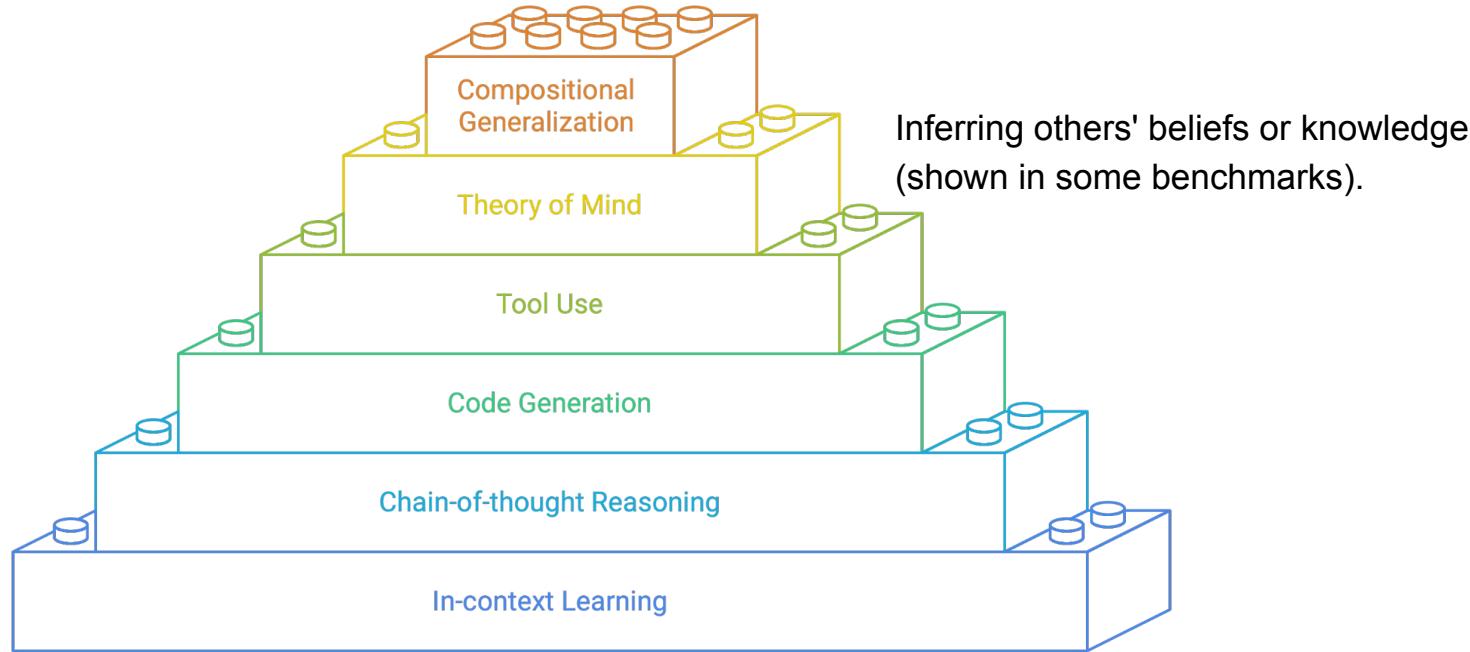


Emergent Behaviours of LLMs



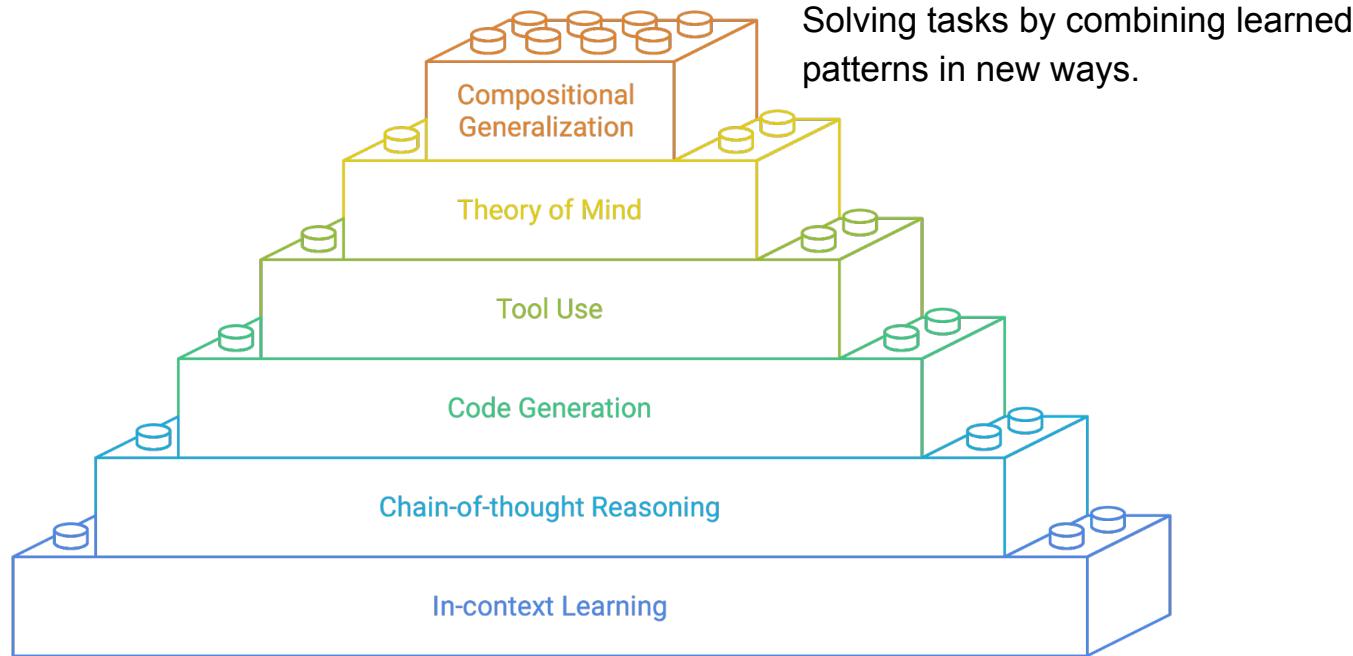


Emergent Behaviours of LLMs





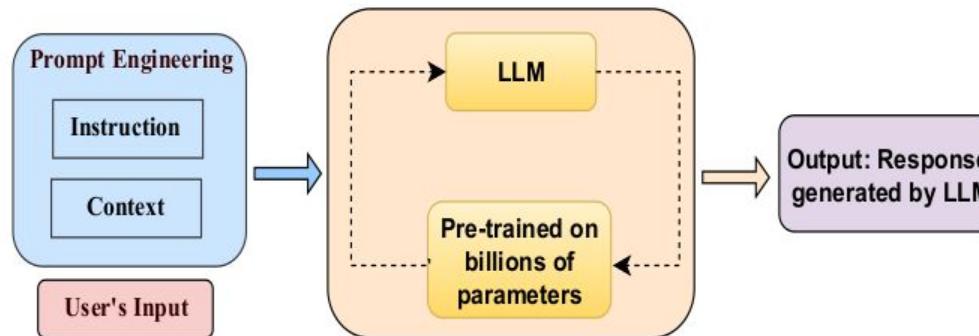
Emergent Behaviours of LLMs





Instructing LLMs

- LLMs were **not originally trained** to follow commands
- This capability emerged from the way that LLMs were **trained**, which we will discover in the next section
- Using this capability, which we now call **prompting** an LLM, is how we communicate with an LLM





Types of Prompting

Few-shot
Prompting

Zero-shot
Prompting

Chain-of-Thoughts
Prompting

Prompt Engineering

Self-Consistency
Prompting

Prompt Chaining
Prompting

Tree of Thoughts
Prompting

Directional Stimulus
Prompting



Types of Prompting

Directly ask the model to perform a task without providing examples.



Zero-shot
Prompting

Few-shot
Prompting

Chain-of-Thoughts
Prompting

Examples: "Translate the following sentence to French:
I love learning about AI."

Self-Consistency
Prompting

Prompt Engineering

Tree of Thoughts
Prompting

Directional Stimulus
Prompting

Prompt Chaining
Prompting



Types of Prompting

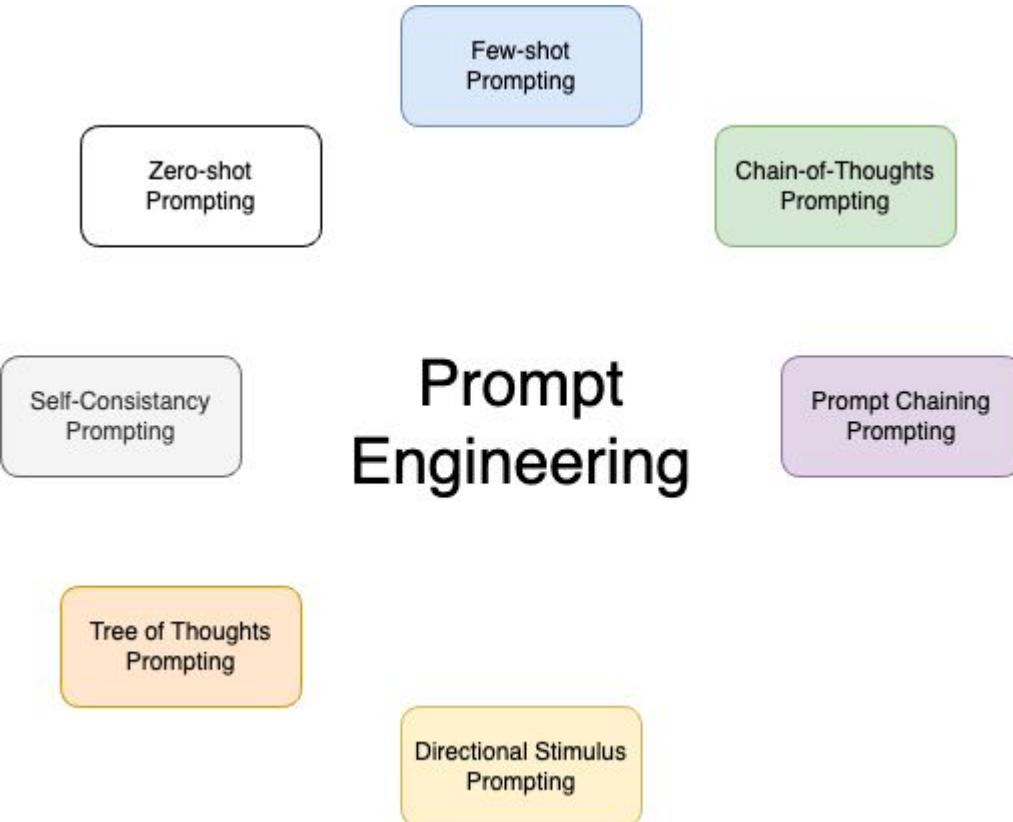
Generate multiple reasoning paths and select the most consistent or common answer.

Example:

"Solve: If a train leaves at 3 PM and travels 60 km/h, how far will it go by 6 PM?"

→ Ask multiple times and aggregate answers like voting.

Prompt Engineering



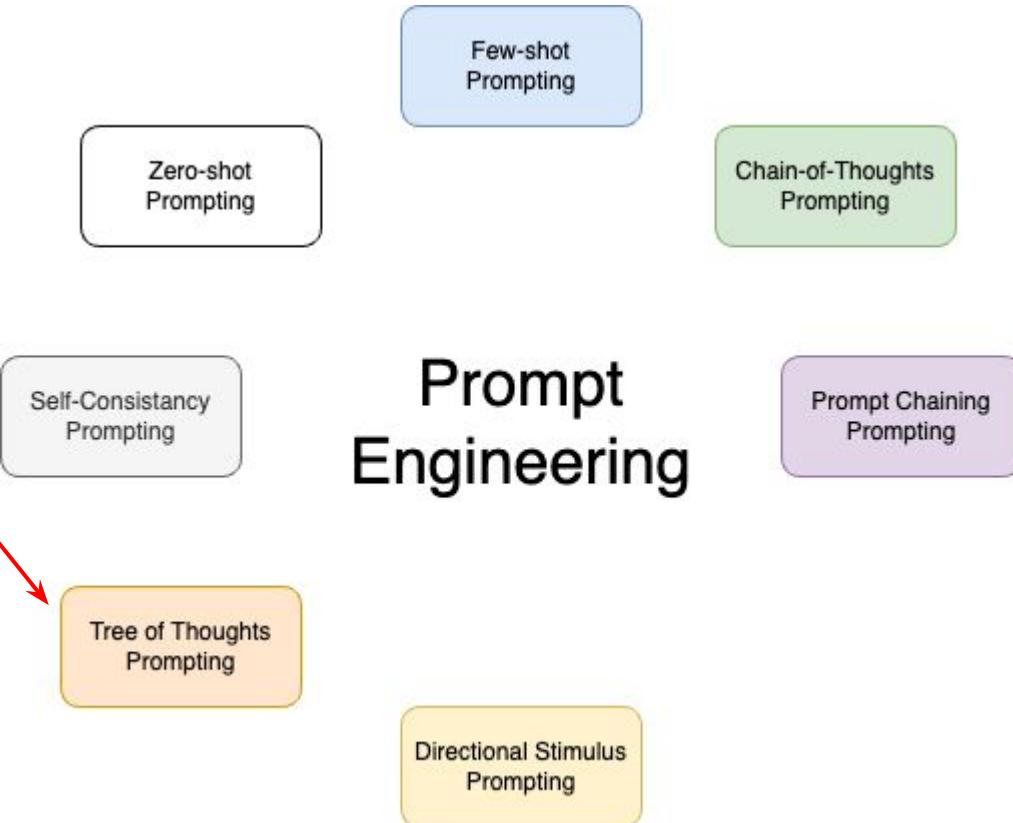


Types of Prompting

Ask the model to explore multiple reasoning paths before selecting the best outcome.

"What are three possible strategies to increase customer engagement on an e-commerce site? Evaluate each and pick the most effective one."

Prompt Engineering





Types of Prompting

Provide a few input-output examples before asking the model to perform the task.

Examples:

Q: What's the capital of France?

A: Paris

Q: What's the capital of Japan?

A: Tokyo

Q: What's the capital of Australia?

A: ?

Few-shot
Prompting

Zero-shot
Prompting

Chain-of-Thoughts
Prompting

Self-Consistency
Prompting

Prompt Chaining
Prompting

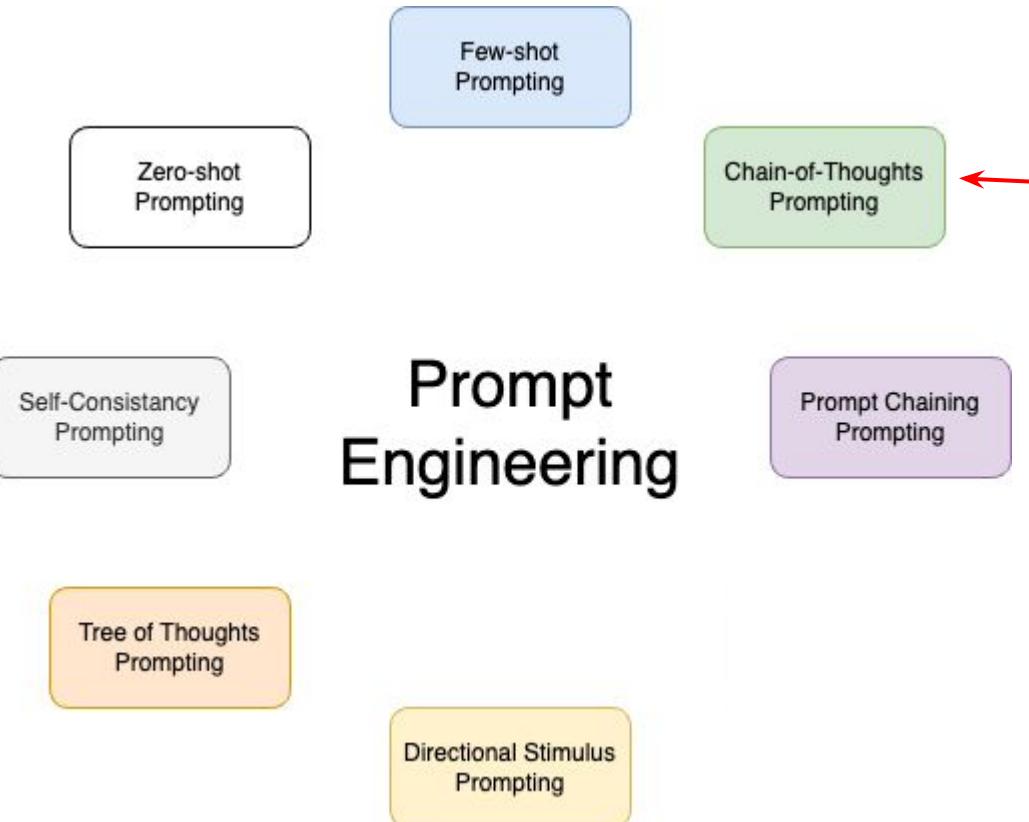
Prompt Engineering

Tree of Thoughts
Prompting

Directional Stimulus
Prompting



Types of Prompting



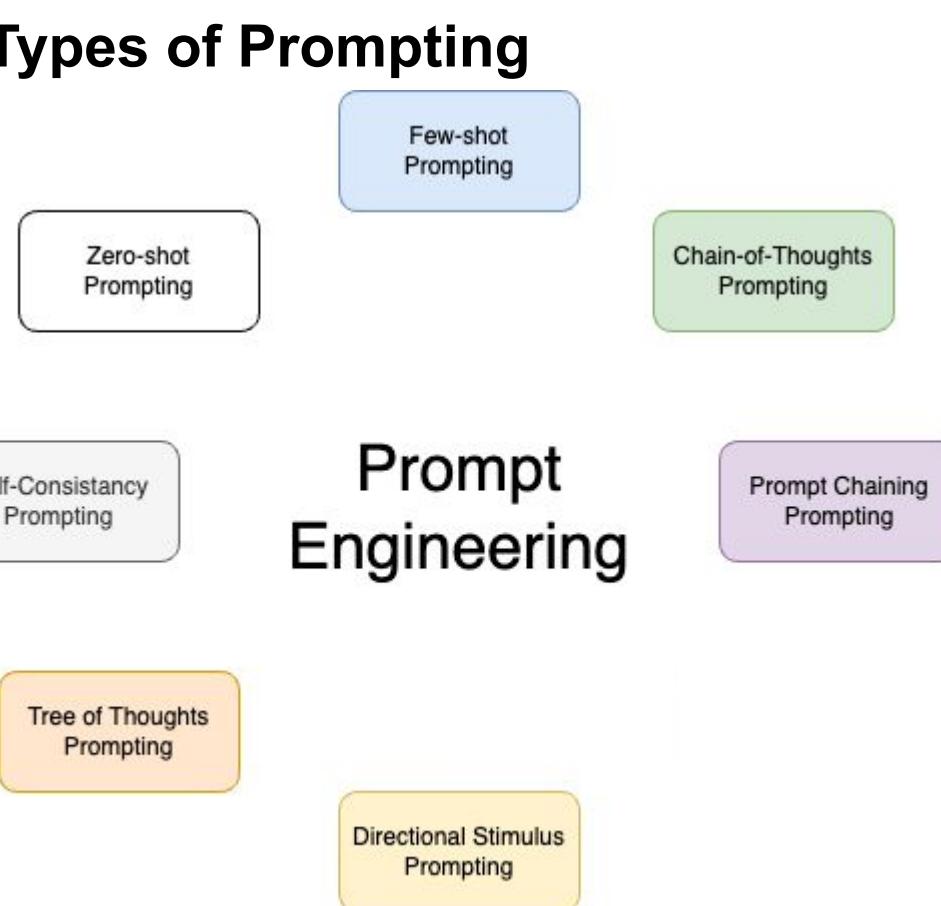
Prompt the model to show intermediate reasoning steps.

"If there are 3 red balls and 2 blue balls in a bag, and you draw one without looking, what is the probability it is red? Let's think step-by-step."



Types of Prompting

Prompt Engineering



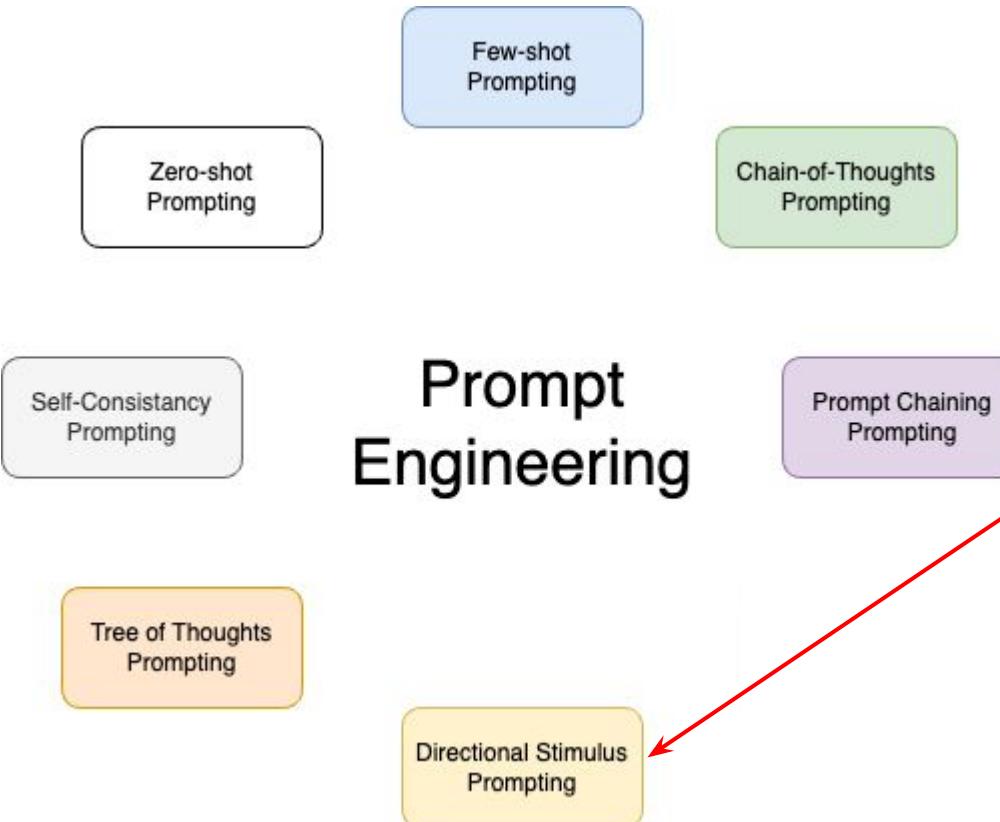
Break down a task into multiple stages, using the output of one prompt as the input to the next.

Example:

1. Prompt 1: "Summarize the following article."
2. Prompt 2: "Now, generate 3 questions based on that summary."
3. Prompt 3: "Answer those questions using the summary."



Types of Prompting



Nudge the model toward a particular style, perspective, or goal by using carefully chosen language.

"You are an optimistic career coach. Give me advice on switching to AI from a finance background."



Hands on: Prompting_Techniques_with_Deepseek.ip ynb

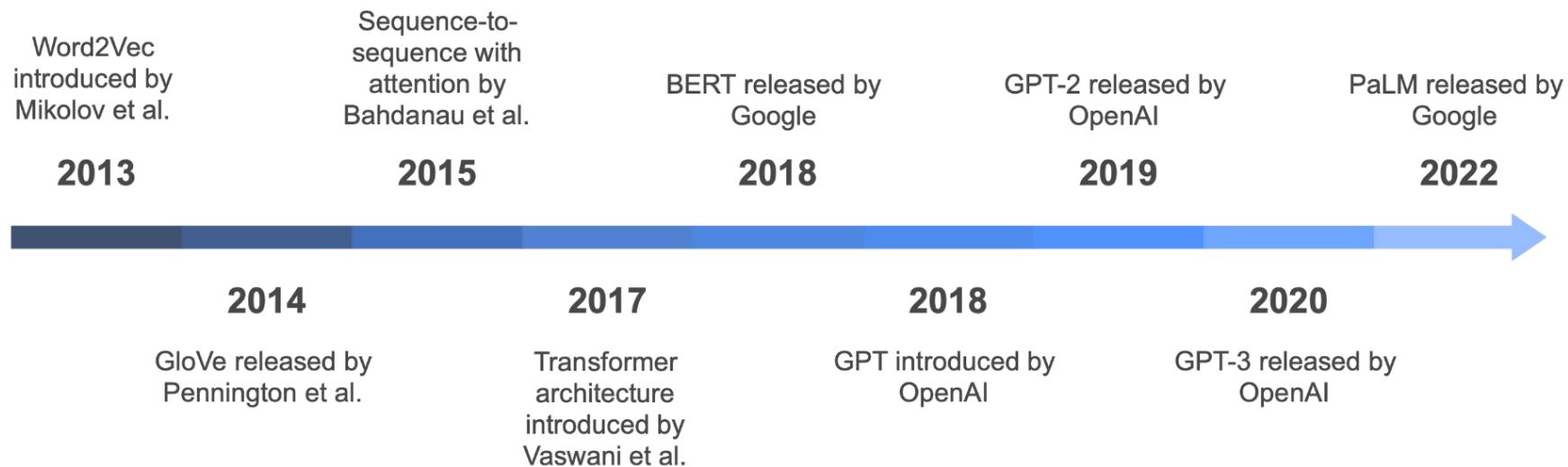


AMERICAN
UNIVERSITY
OF BEIRUT

We've seen a lot of LLMs,
but how did they come to
be?



LLM Timeline Part 1





LLM Timeline Part 2

Meta LLaMA 1 Release	Anthropic Claude 2 Release	OpenAI ChatGPT Adoption	Meta LLaMA 3 Release	Meta LLaMA 3.1 Release	Anthropic Claude 3.7 Sonnet Release	Anthropic Claude 4 Anticipation
Meta introduces LLaMA for research use	Anthropic releases Claude 2 as a GPT-4 competitor	ChatGPT gains mass adoption with millions of users	Meta launches LLaMA 3 with enhanced performance	Meta adds a 405B-parameter variant to LLaMA 3	Anthropic introduces Claude 3.7 Sonnet with hybrid reasoning	Industry anticipates the release of Claude 4



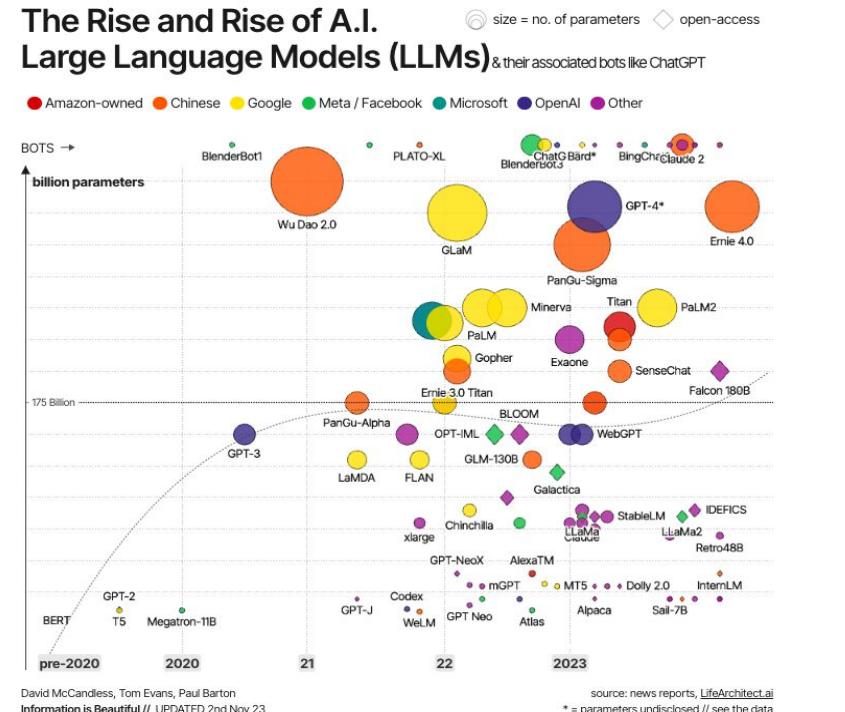
OpenAI GPT-4 Launch	Meta LLaMA 2 Release	DeepMind Gemma 1 Release	Anthropic Claude 3.5 Sonnet Release	Mistral AI Releases	DeepMind Gemma 3 Release
OpenAI launches GPT-4 with multimodal capabilities	Meta releases LLaMA 2 under a permissive license	DeepMind introduces Gemma 1 for on-device use	Anthropic releases Claude 3.5 Sonnet with vision reasoning	Mistral AI launches models targeting efficiency	DeepMind launches Gemma 3 for therapeutics

What is the difference between 2023 and 2025 LLMs? What is changing?



LLM Advancements

- Well, lots of things are changing like **architecture, size, training data**.
- One of the most prominent advancements in LLM generation is **size** of the model.
- The last model that openai released its size was GPT 4 with roughly **1.8 Trillion parameters**.





AMERICAN
UNIVERSITY
OF BEIRUT

Not all of these LLMs' architectures are a secret.. We have open-source LLMs!



Open Source Vs Closed Source LLMs

Open-Source LLMs

Offers transparency, customizability, and community support but may lack professional support and security.

Closed-Source LLMs

Provides performance, security, and vendor support but at higher costs and less flexibility.



Open Source vs Closed Source LLMs

Type\Functionality	Open Source	Closed Source
Use Case	Sufficient for general-purpose tasks	Specialized or enterprise-level applications might benefit from the performance and support of closed-source models.
Security	Open-source models require careful evaluation of security risks,	Closed-source models offer more control over access and data.
Cost	Open-source models can be more cost-effective for personal projects and experimentation, but would be more expensive to self-host on cloud solution platforms.	On an enterprise level, closed-source models often involve licensing fees, which makes them more expensive.
Customization	Open-source models offer greater flexibility for customization	Limited and pre-defined capabilities for customization.
Support	Open-source models rely on community support, which may not always be as reliable.	Dedicated and fast professional customer support and bug fixing for any issues.



Using a Closed Source LLM

- **Cohere**, the company behind the frontier LLMs Command R and Command A, has a '**Free Trial API Key**' that you can generate or use without adding a payment method.
- You can also use the Cohere playground to **experiment** with the available models using a UI

Trial keys FREE

Calls made using Trial keys are free of charge. Trial keys are rate-limited, and cannot be used for commercial purposes.



Using a closed source LLM

Using Cohere is completely **beginner friendly** since the playground also generates an example API usage for you. This saves you reading through documentation.

View Code

python

```
from cohere import ClientV2

client = ClientV2(api_key=<YOUR_API_KEY>)
client.chat_stream(
    model="command-a-03-2025",
    messages=[
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": "hello"
                }
            ]
        },
        {
            "role": "assistant",
            "content": "Hello! How can I assist you today?"
        }
    ],
    ...
```

Close



Using an open source LLM

- To get started using an **open source LLM**, you can browse and select an LLM on hugging face.
- The popular models, like meta's latest model, llama 4, have tutorials on using the model

```
# Use a pipeline as a high-level helper
from transformers import pipeline

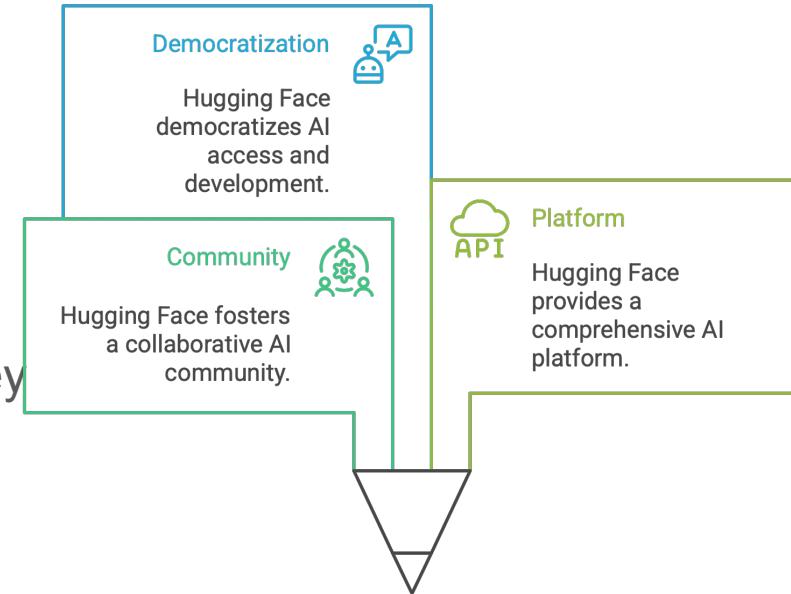
pipe = pipeline("image-text-to-text", model="meta-llama/Llama-4-Scout-17B-16E-Instruct")
messages = [
    {
        "role": "user",
        "content": [
            {"type": "image", "url": "https://huggingface.co/datasets/huggingface/documentation-images/resolve/main/p-blog/candy.JPG"},
            {"type": "text", "text": "What animal is on the candy?"}
        ]
    },
]
pipe(text=messages)
```



The Home of Open Source: HuggingFace

- Hugging Face has become a **central hub** in the world of **Artificial Intelligence**, particularly in **Natural Language Processing (NLP)**.
- It is more than just a company; it's a **community**, a **platform**, and a **driving force** behind the democratization of AI.
- We will explore what Hugging Face is, its key **components**, and its **impact** on the AI landscape

Hugging Face's AI Impact





The Home of Open Source: HuggingFace

- Hugging Face has **community & model hub** with 2M+ models.
- It is built around several **core components** that contribute to its mission of making good machine learning accessible.
- These components are **several libraries** that are free for us to use:

hub, datasets, transformers, peft, trl, accelerate



HuggingFace Hub

What it is: Central repository for sharing and discovering models, datasets, spaces (apps).

Key features:

- Model cards & documentation
- Versioning & lineage
- Inference endpoints (host your model with a few clicks)

The screenshot shows the Hugging Face Hub interface. At the top, there's a search bar with placeholder text "Search models, datasets, users...". Below the search bar are navigation tabs: Main, Tasks, Libraries, Languages, Licenses, and Other. Under the "Tasks" section, there are buttons for Text Generation, Any-to-Any, Image-Text-to-Text, Image-to-Text, Image-to-Image, Text-to-Image, Text-to-Video, and Text-to-Speech, with a "+ 43" button indicating more options. The "Parameters" section has a slider from <1B to >500B. The "Libraries" section includes PyTorch, TensorFlow, JAX, Transformers, Diffusers, Safetensors, ONNX, GGUf, Transformers.js, MLX, MLX, Keras, and a "+ 40" button. The "Apps" section lists vLLM, TGI, llama.cpp, MLX LM, LM Studio, Ollama, Jan, and + 12. On the right, a sidebar shows a list of models with their names, descriptions, and statistics: moonshotai/Kimi-K2-Instruct (Text Generation, updated 10 hours ago, 25.1k, 1.13k), HuggingFaceTB/SmollM3-3B (Text Generation, updated 5 days ago, 59.5k, 469), THUDM/GLM-4.1V-9B-Thinking (Image-Text-to-Text, updated 7 days ago, 38.2k, 604), black-forest-labs/FLUX.1-Kontext-dev (Image-to-Image, updated 18 days ago, 274k, 1.66k), mstralai/Devstral-Small-2507 (Text Generation, updated 5 days ago, 9.58k, 215), moonshotai/Kimi-K2-Base (Text Generation, updated 3 days ago, 2.07k, 201), LiquidAI/LFM2-1.2B (Text Generation, updated about 22 hours ago, 4.02k, 158), and microsoft/Phi-4-mini-flash-reasoning (Text Generation, updated about 11 hours ago, 2.05k, 141).



HuggingFace Datasets

What it is: A library to easily load, process, and share datasets.

Key features:

- On-the-fly streaming of large datasets
- Built-in support for common benchmarks (GLUE, SQuAD, etc.)
- Efficient caching, splits, and metadata

The screenshot shows the HuggingFace Datasets interface. At the top, there are tabs for Main, Tasks, Libraries, Languages, and Licenses. Below these are sections for Other, Modalities (with categories like 3D, Audio, Document, Geospatial, Image, Tabular, Text, Time-series, and Video), Size (rows) with a slider from <1K to >1T, and Format (with options for json, csv, parquet, imagefolder, soundfolder, webdataset, text, and arrow). To the right, a search bar shows 'Datasets 449,845' and a 'Filter by name' button. Below the search bar, there is a list of dataset entries, each with a thumbnail, the dataset name, a viewer link, the last update time, the number of rows, the size, and the number of likes. Some examples include 'fka/awesome-chatgpt-prompts', 'hackaprompt/Pliny_HackAPrompt_Dataset', 'NousResearch/Hermes-3-Dataset', 'MrSupW/ContextASR-Bench', 'microsoft/rStar-Coder', 'XenArcAI/MathX-5M', and 'HuggingFaceFW/fineweb-2'.



HuggingFace Transformers library

Hugging Face Transformers is an **open-source** Python library that provides **access** to thousands of **pre-trained Transformers** models

What it is: The flagship library for using pre-trained transformer models.

Key features:

- 1000s of encoder/decoder/decoder-only architectures
- Tokenizers, pipelines, and model zoo
- Seamless switch between PyTorch, TensorFlow, JAX

```
from transformers import TrainingArguments
```

```
training_args = TrainingArguments(  
    output_dir="distilbert-rotten-tomatoes",  
    learning_rate=2e-5,  
    per_device_train_batch_size=8,  
    per_device_eval_batch_size=8,  
    num_train_epochs=2,  
    push_to_hub=True,  
)
```

```
from transformers import Trainer
```

```
trainer = Trainer(  
    model=model,  
    args=training_args,  
    train_dataset=dataset["train"],  
    eval_dataset=dataset["test"],  
    tokenizer=tokenizer,  
    data_collator=data_collator,  
)
```

```
trainer.train()
```



HuggingFace PEFT Library

What it is: Techniques for adapting large models with minimal new parameters.

Key methods:

- LoRA (Low-Rank Adaptation)
- Adapters, Prefix-Tuning, BitFit

Why it matters:

- Save GPU memory & training time
- Fine-tune 100B+ models on a single GPU

- `task_type`: the task to train for (sequence-to-sequence language modeling in this case)
- `inference_mode`: whether you're using the model for inference or not
- `r`: the dimension of the low-rank matrices
- `lora_alpha`: the scaling factor for the low-rank matrices
- `lora_dropout`: the dropout probability of the LoRA layers



AMERICAN
UNIVERSITY
OF BEIRUT

Hands on: 2_huggingface_libraries.ipynb



**Does the entire model need to released?
Can we have access to training the model
without its architecture?**



Open Source Vs Open Weights

Feature	Open Source	Open Weights
Definition	The codebase (e.g., model architecture, training scripts) is publicly available, modifiable, and often licensed under an open-source license like Apache 2.0 or MIT.	The trained model weights (parameters) are publicly released, allowing you to run or fine-tune the model. The code may or may not be open.
Can view/modify code?	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> Not necessarily (only weights shared, not training code)
Can retrain from scratch?	<input checked="" type="checkbox"/> Yes (if data is also available)	<input type="checkbox"/> No (you can only use the existing weights)
License Type	Usually under recognized OSI-approved licenses	Often under custom, non-commercial, or research-only licenses
Example	<ul style="list-style-type: none">◆ Mistral, Dolly, Pythia, Falcon (with full code + weights)◆ GitHub repositories with training code, tokenizer logic, and scripts	<ul style="list-style-type: none">◆ LLaMA 2, Gemma, Command-R+, which release weights (with usage restrictions) but not full training code
Can use for commercial applications?	<input checked="" type="checkbox"/> Yes (if license allows, like Apache 2.0)	<input type="checkbox"/> Often not allowed—depends on terms (e.g., LLaMA 2's restricted license)



Open Source vs Open Weights

- Some LLM providers, like openai, allow **fine-tuning** their models via a **closed-box API** without open sourcing their weights.

Term	Can run locally	Can fine-tune offline	Access to weights	Access to code	License
Open weights (e.g. LLaMA 2)	✓ Yes	✓ Yes (with hardware)	✓ Yes	✗ No	🟡 Restricted
Training playground (e.g. OpenAI)	✗ No	✓ Yes (via API)	✗ No	✗ No	🔒 Fully closed
Fully open-source (e.g. Mistral)	✓ Yes	✓ Yes	✓ Yes	✓ Yes	🟢 Permissive



Differences Between LLMs

- Now that we see a surge of new LLMs in the market, what is the difference between them? Do we really need that many?

Openai Models	Anthropic Models	Google Models
<p>Excel in general knowledge and creative tasks.</p> <ul style="list-style-type: none">Strong reasoning capabilitiesEffective summarizationVersatile content generation	<p>Superior in code generation and technical reasoning.</p> <ul style="list-style-type: none">Advanced coding assistanceRobust safety alignmentNuanced instruction following	<p>Strong in factual knowledge and multimodal capabilities.</p> <ul style="list-style-type: none">Extensive factual databaseResearch-oriented responsesIntegrated search functionality

Difference Between Frontier Models

Let's take an example to test these models' reasoning and explanatory abilities.

Prompt:

"A snail is at the bottom of a 10 m well. Each day it climbs up 3 m, but each night it slides back 2 m. How many days does it take for the snail to escape? Briefly explain your reasoning."



Difference Between Frontier Models

o4-mini-high

Thought for a few seconds >

The snail makes a net gain of 1 m each full day (3 m up minus 2 m down). On the morning of the day it finally escapes it won't slide back.

- After 7 full days it has climbed $7 \times 1 = 7$ m (end of night 7).
- On day 8 it climbs 3 m from 7 m up to 10 m and gets out.

So it takes **8 days** for the snail to escape.



Difference Between Frontier Models

Claude sonnet 4

The snail takes **8 days** to escape the well.

Here's the reasoning: Each full day-night cycle, the snail makes a net progress of 1 meter (climbs 3m during the day, slides back 2m at night). However, we need to consider what happens on the final day.

After 7 complete day-night cycles, the snail will be at 7 meters. On the 8th day, when it climbs 3 meters, it reaches 10 meters and escapes the well - so it doesn't slide back that night.

The key insight is that once the snail reaches the top during a day, it's out and won't slide back, so we can't just divide 10 by the net daily progress of 1 meter.



Difference Between Frontier Models

Gemini 2.5 Pro



Show thinking ^

Estimating the Snail's Climb

I'm starting by calculating the snail's daily height gain, accounting for the slide. Then, the focus shifts to figuring out how many days it takes for the climb to exceed the bounds of the 10-meter post. This should give a solid basis for solving the problem.

Clarifying the Climb

I've refined the daily gain calculation, emphasizing the net ascent. After that, I'm zeroing in on the "breakout point" – the crucial day where the snail's climb puts it over the top, no matter the slide. This point will then be incorporated with the daily gain in the final answer.

On day 8, the snail will escape the well.

Here's the reasoning:

The snail makes a net daily progress of 1 meter (3 meters up - 2 meters down). By the morning of the 7th day, the snail will have reached a height of 7 meters. On the 8th day, the snail climbs its usual 3 meters. a height of 10 meters, and escapes the well without sliding back down that night.



Difference Between Frontier Models

Cohere Command A

To determine how many days it takes for the snail to escape the 10-meter well, let's analyze its daily progress:

1. **Daily Net Progress:** Each day, the snail climbs 3 meters, but each night it slides back 2 meters. Therefore, the net progress each day is:

$$3 \text{ m} - 2 \text{ m} = 1 \text{ m}$$

2. **Progress Over Multiple Days:** After 7 days, the snail will have climbed a net distance of:

$$7 \text{ days} \times 1 \text{ m/day} = 7 \text{ m}$$

3. **Final Day's Climb:** On the 8th day, the snail starts at 7 meters. It climbs 3 meters during the day, reaching:

$$7 \text{ m} + 3 \text{ m} = 10 \text{ m}$$

Since the snail reaches the top of the well during the day on the 8th day, it does not slide back that night. Therefore, the snail escapes the well on the 8th day.

The answer is:



Hands on: Create_a_chat_interface_with_gradio.ipyn b



AMERICAN
UNIVERSITY
OF BEIRUT

LLM Architecture



Differences Between LLMs

In the previous slides, we saw **differences** in capabilities between OpenAI, Anthropic, and Google models. Let's explore *why*.

Factor	OpenAI (GPT)	Anthropic (Claude)	Google (Gemini)
Data Diversity	Broad, creative web sources	High-quality, technical data	Factual, multimodal, research
Instruction Tuning	RLHF + user feedback loops	Constitutional AI	Research-tuned and factuality
Architecture Tweaks	Dense, multimodal adapters	Possibly MoE + long context	MoE, native multimodal
Strengths	Creativity, summarization	Code, alignment, reasoning	Factual Q&A, image-text tasks



Differences between LLMs

Even with similar transformer foundations, these LLMs evolve differently because of unique design choices in **architecture, pre-training data, tuning strategies, and intended use cases**. The result is a set of models that are each "best-in-class" in different domains.

Let's cover each of these design choices.

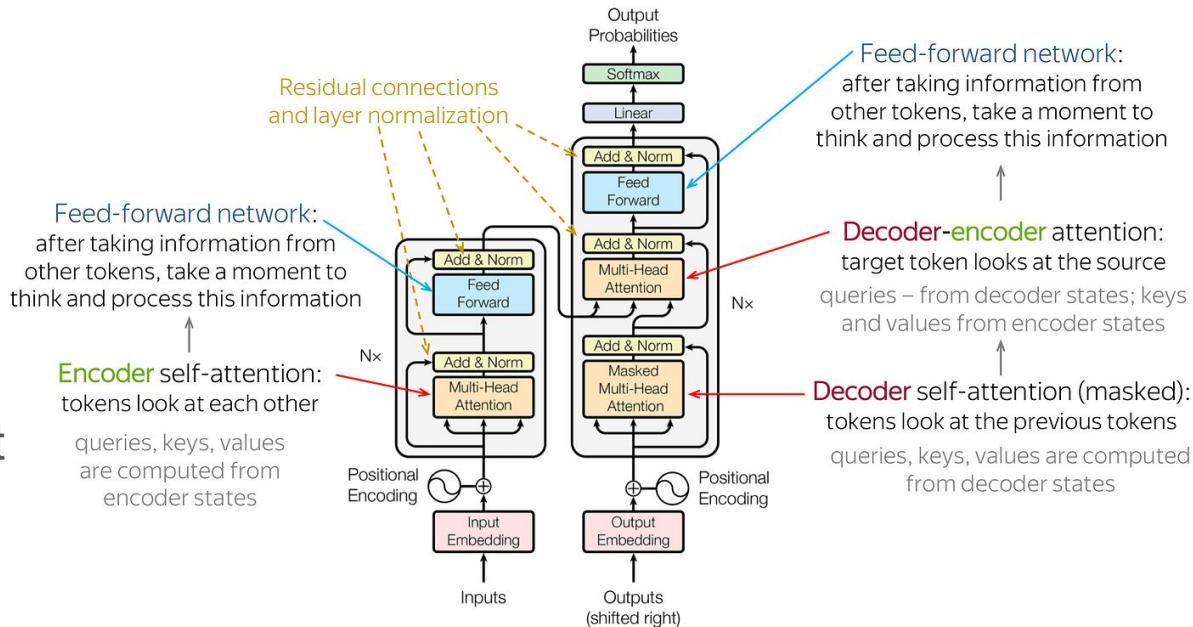


Encoder-Decoder Models

Combines both encoder and decoder for input → output tasks like translation and summarization.

Description:

- **Encoder** processes full input context.
- **Decoder** generates output based on encoder's representation + previous output tokens





Encoder-Decoder Vs Decoder-Only Models

In the previous slide, we mentioned that encoder-decoder architecture excels when the input **is completely different** from the output. However, we saw that decoder-only models like the GPTs and Claude actually perform incredibly well in these tasks as well.

Does this mean that decoder-only models are sufficient?

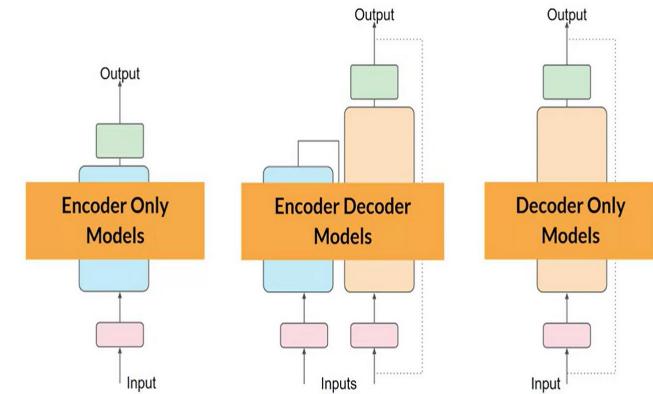
Is this the end of encoder-decoder models?



Encoder-Decoder Vs Decoder-Only Models

GPT-style models **blurred the boundary** between the capabilities of different architectures because of:

- **Massive scale:** With billions/trillions of parameters, they learn internal representations *on the fly*, without needing a separate encoder.
- **Causal attention + long context:** They handle input and output in one unified sequence.
- **Flexible prompting:** You provide the input + instructions in a single prompt — and the model “implicitly” encodes the input before generating.

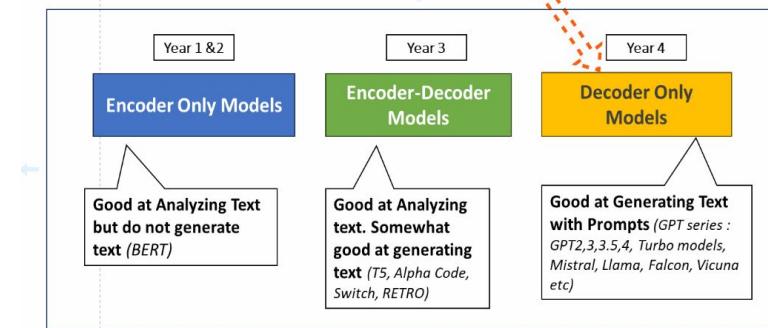




Encoder-Decoder Vs Decoder-Only Models

However, Encoder-Decoder models are not completely obsolete.

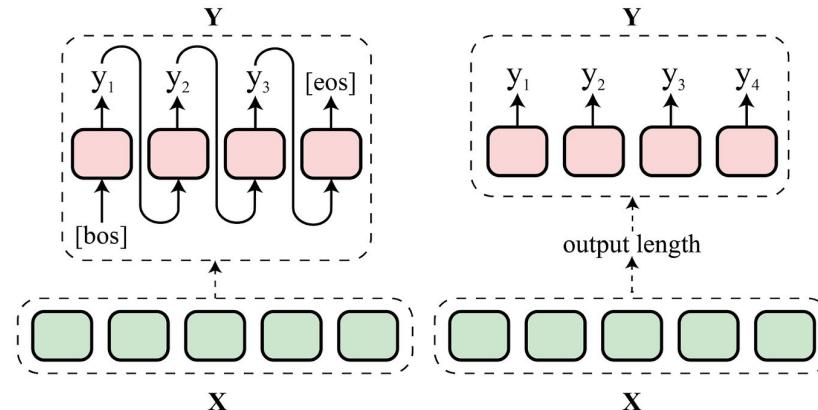
- For general-purpose, large-scale LLMs, decoder-only has become the dominant paradigm.
- For smaller, task-specific models, encoder-decoder can still outperform:
 - T5 and BART are more efficient for tasks like summarization or classification in low-resource or on-device settings.
 - In multilingual settings or low-data regimes, encoder-decoder may generalize better.





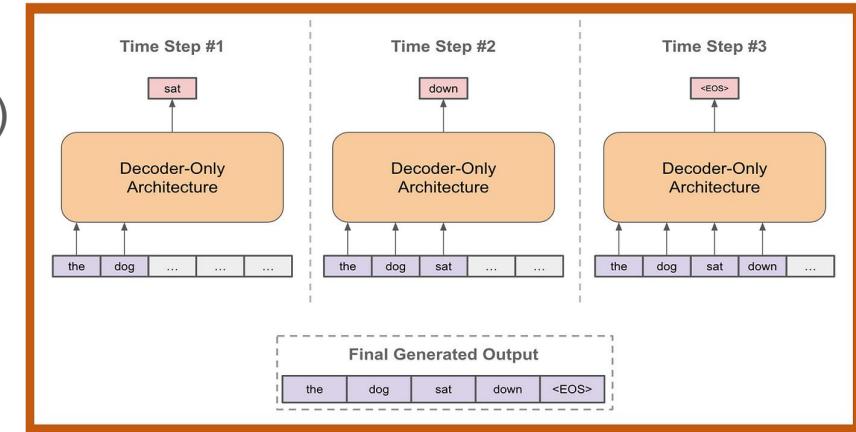
Decoder-Only Models

These are the most common for **autoregressive text generation** (e.g., GPT family). Autoregressive text generation is a method where a language model predicts the next word in a sequence based on the words that have already been generated. It works by iteratively predicting one word at a time, building upon the previously generated text.



Decoder-Only Models

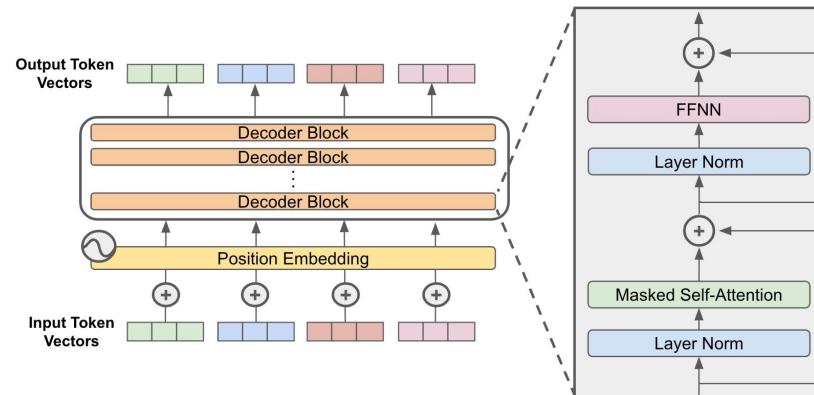
- **Description:** Uses only the *decoder* part of the Transformer (causal attention – attends to past tokens only).
- **Strengths:**
 - Excellent for **generation** tasks
 - Optimized for left-to-right token prediction.
- **Examples:**
 - GPT-2, GPT-3, GPT-4 (OpenAI),
 - Claude (Anthropic) (Closed Source)
 - LLaMA, Mistral, Falcon
 - Phi (open source)





Decoder-Only Models

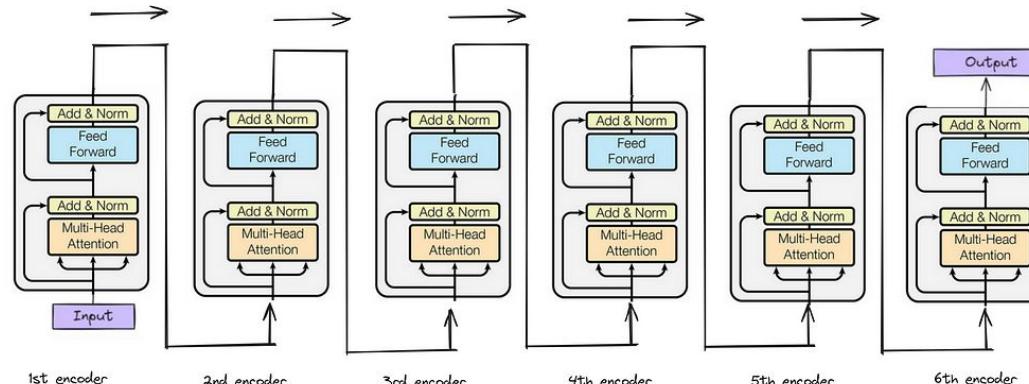
- Decoder-only models work **without explicit encoding** because they are designed for **autoregressive text generation**, where they predict the next token based on the preceding ones, effectively encoding the input implicitly while generating the output.
- This contrasts with encoder-decoder models that first encode the input into a fixed-length representation before decoding





Encoder-Only Models

- Encoder-only models, like BERT, use the **encoder part** of the Transformer architecture to process input text and create a **contextualized representation**.
- They don't generate new text, but instead focus on **understanding the input** and are well-suited for tasks like text classification, sentiment analysis, and named entity recognition.

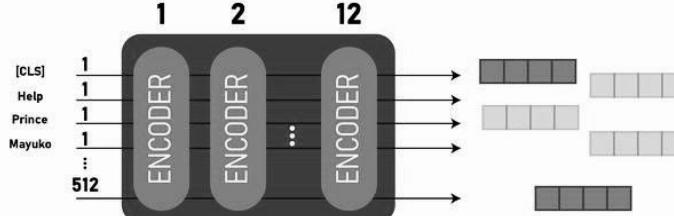




Encoder-Only Models

Optimized for understanding and classification, not generation.

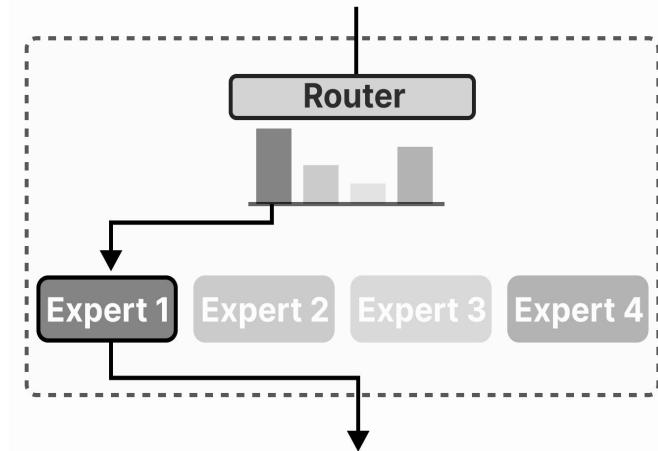
- **Description:** Uses only the *encoder* side (bidirectional attention – attends to all tokens in input).
- **Strengths:**
 - Ideal for **text classification, retrieval, sentence embeddings**.
 - Understands context more holistically.
- **Examples:**
 - BERT (Google), RoBERTa (Meta), DistilBERT (Hugging Face), DeBERTa (Microsoft)



Mixture of Experts (MoE) Models

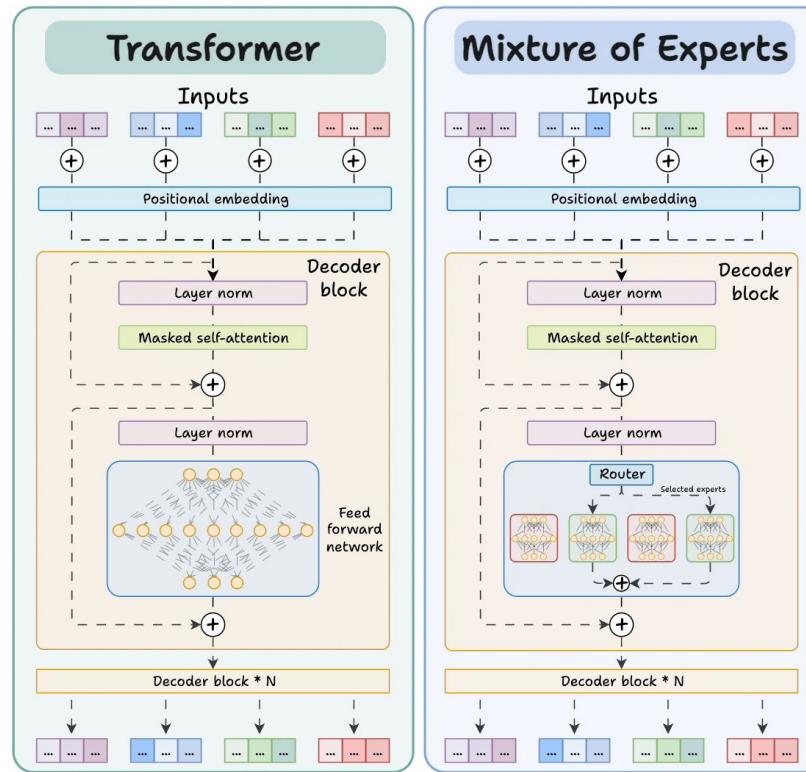
An MoE model is a type of large language model that utilizes a neural network architecture where the model is divided into multiple specialized subnetworks, called "experts". Not every layer/expert is used during each forward pass → efficiency & specialization.

- **Strengths:**
 - Better scaling with less computational cost.
 - Enables **expert specialization**
- **Examples:**
 - GShard, Switch Transformer (Google)
 - Mixtral (Mistral)





Mixture of Experts (MoE) Models





MoE core components

Core Components

1. Experts

- Independent feed-forward sub-networks (typically two-layer MLPs).
- All experts share the same input/output dimensions but have separate weights.

2. Gating Network

- A small softmax classifier that, for each token (or token-group), computes routing scores over all experts.
- Selects top-k experts (often k=1 or 2) to handle that token.

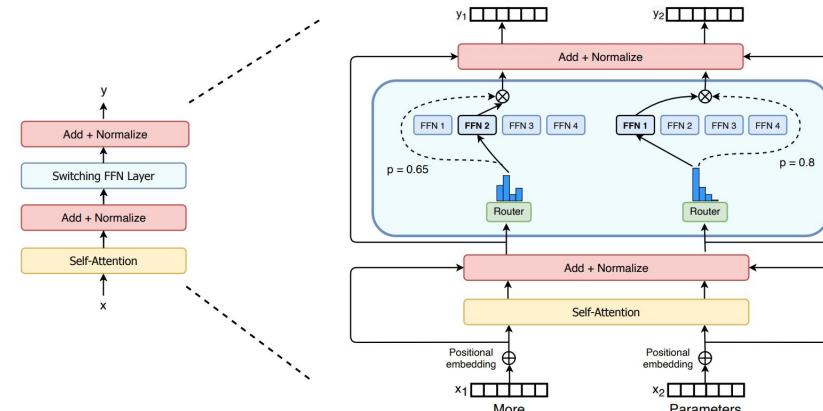
3. Sparse Routing

- Only the selected experts receive and process the token's hidden state.
- Outputs from experts are weighted (by gate scores) and summed back into the main model stream.



MoE Forward Pass

1. **Input Hidden State** → sent into the **Gating Network**.
2. **Gate** produces scores $g_1 \dots g_N$ for N experts.
3. **Top-k Selection**: choose experts with highest g_i .
4. **Dispatch** the hidden state to those experts → each applies its expert-MLP.
5. **Combine** expert outputs
6. **Continue** through remaining transformer layers.





Multi-modal LLMs

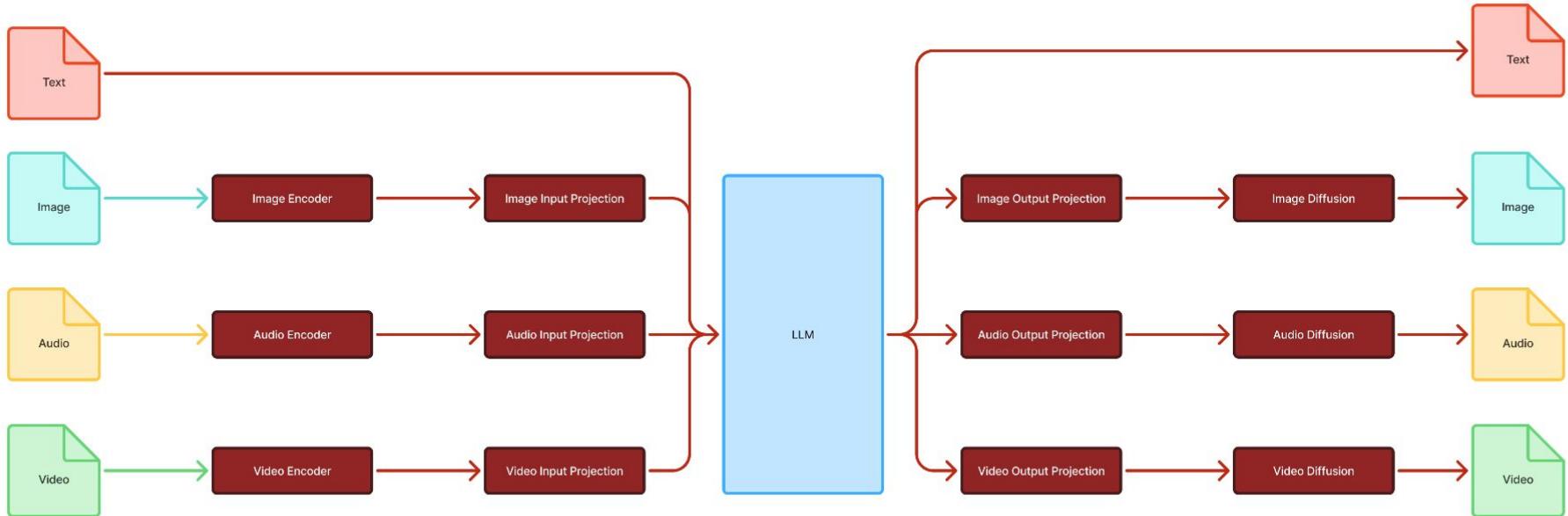
Can process multiple data types (text + image + audio).

- **Description:**
 - Modified architecture with cross-modal embeddings or separate modality encoders.
- **Strengths:**
 - Useful for **vision-language**, **video Q&A**, and **robotic control**.





Multi-modal LLMs





AMERICAN
UNIVERSITY
OF BEIRUT

Hands on: **multimodal_models_with_openai.ipynb**



Recently, we saw a new type of model emerge: Reasoning Models

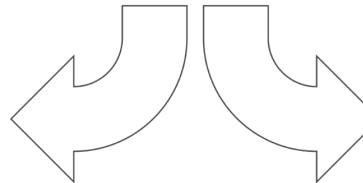


Reasoning vs Non-reasoning models

Have you ever wondered what is the difference between openai's GPT 4 series models and the o series models? [Previously] Why does one say that it's 'thinking' before we get an output? Why do the thinking times change between different queries? How does the model know how much it needs to think?

O-series models are called 'reasoning' models.

Which type of AI model to develop for specific tasks?



Non-Reasoning Models

Suitable for tasks requiring fluent, contextually plausible outputs based on statistical correlations.

Reasoning Models

Ideal for tasks involving multi-step inference, planning, and logical deduction.

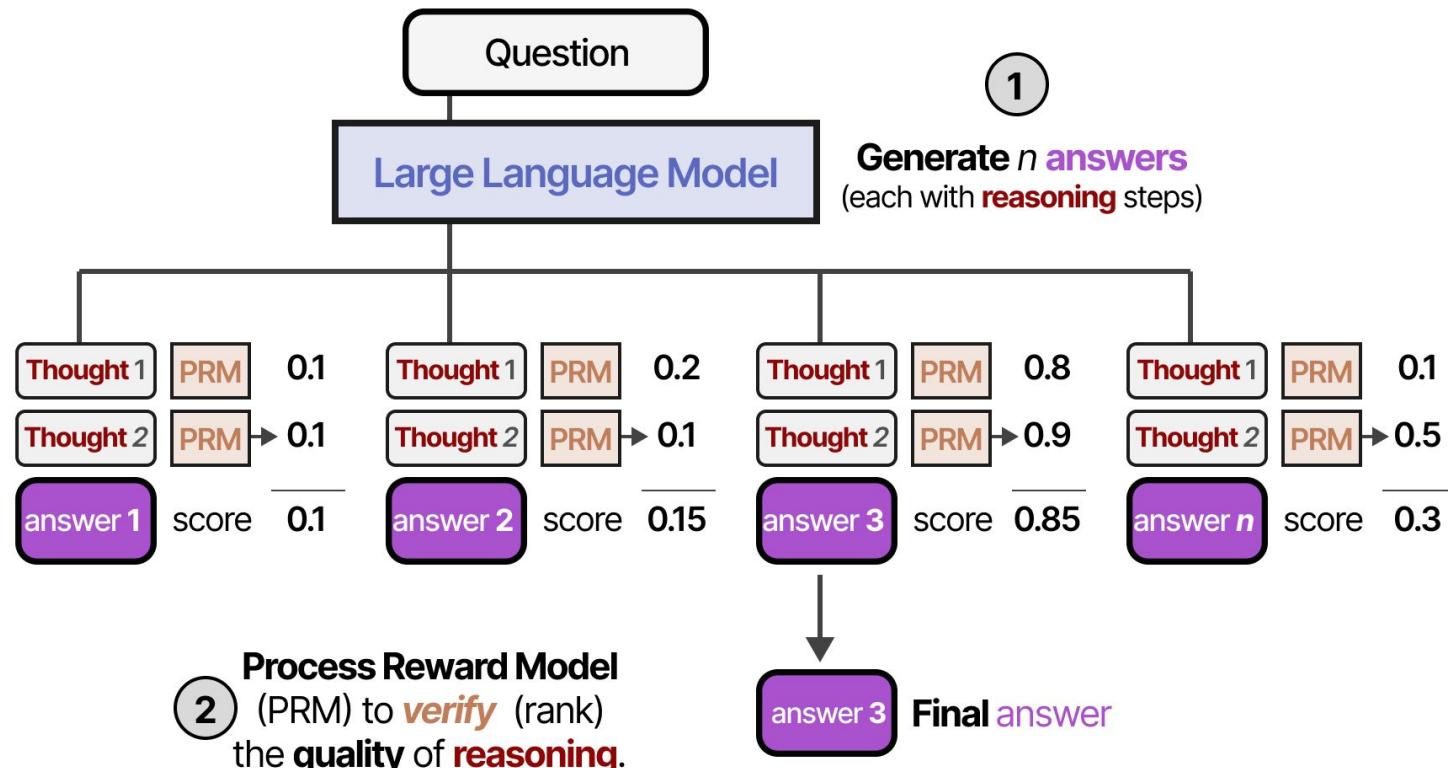


Reasoning vs Non-Reasoning Models

Aspect	Non-Reasoning Models	Reasoning Models
Inference	One-shot next-token prediction	Multi-step, often with intermediate “thought” tokens
Transparency	Opaque—no explicit trace of how answer was reached	Traceable via chain-of-thought or generated sub-program
Generalization	Limited: memorization + shallow pattern match	Better on novel combinations and unseen problem sizes
Architectural Changes	Standard transformer	May include scratchpad heads, program-generation modules, or external tool calls
Training Signals	Language modeling (predict token)	#ERROR!
Typical Failure Modes	Hallucinations, arithmetic errors, logical jumps	Fails when chain-of-thought goes astray; slower
Use Cases	Content generation, summarization, translation	Math problems, code synthesis, multi-hop QA, planning



Reasoning vs Non-Reasoning LLMs





Hands on: reasoning_vs_non_reasoning_models.ipynb



An integral part of an LLM architecture is its size. We've seen newer and larger models able to process thousands of tokens in one input.

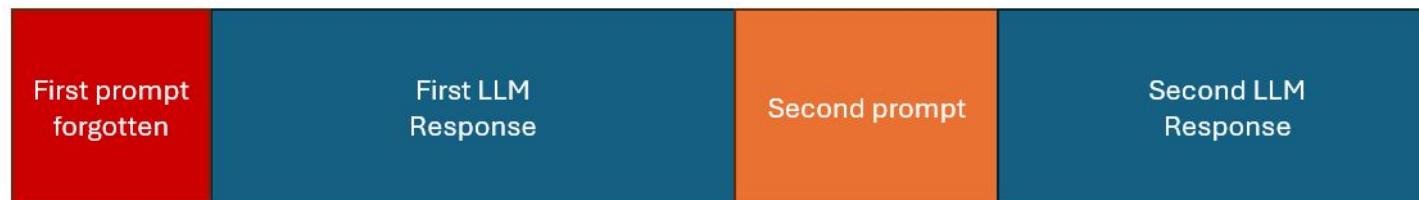


LLM Context Window

The size of an LLM does not only dictate how much information **it can retain**, but also how much information **it can process in one input**.



Second interaction maxes out the available context length and ‘pushes’ the first words out of the LLM’s memory



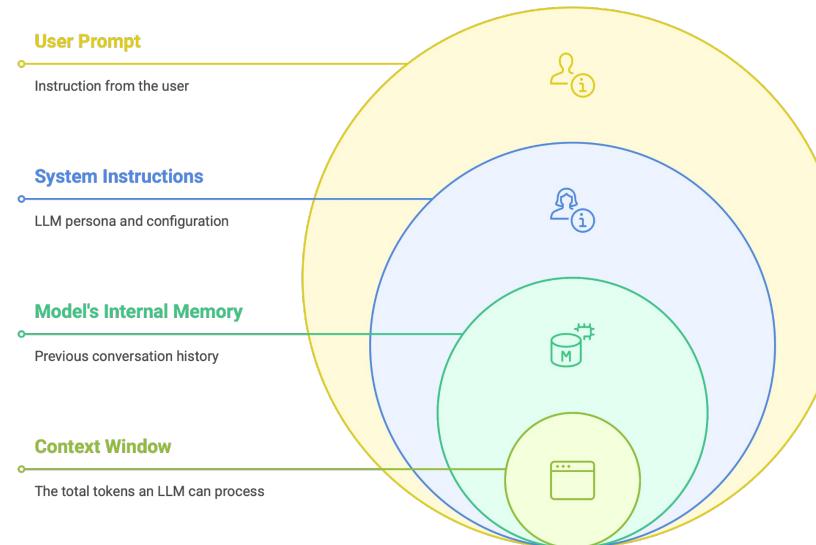


LLM Context Window

How do LLMs handle *longer conversations*, *long documents*, and *reasoning across multiple contexts*?

The **context window** is the maximum number of tokens a model can process in one input sequence to generate the next token.

It includes:





LLM Context Window

Models	Context Window	Input Cost / 1M tokens	Output Cost / 1M tokens	Speed (tokens/second)	Latency
Claude 4 Opus	200,000	\$15	\$75	n/a	1.95 seconds
Claude 4 Sonnet	200,000	\$3	\$15	n/a	1.9 seconds
Gemini 2.5 Flash	1,000,000	\$0.15	\$0.6	200 t/s	0.35 seconds
OpenAI o3	200,000	\$10	\$40	94 t/s	28 seconds
OpenAI o4-mini	200,000	\$1.1	\$4.4	135 t/s	35.3 seconds
GPT-4.1 nano	1,000,000	\$0.1	\$0.4	n/a	n/a
GPT-4.1 mini	1,000,000	\$0.4	\$1.6	n/a	n/a
GPT-4.1	1,000,000	\$2	\$8	n/a	n/a
Llama 4 Scout	10,000,000	\$0.11	\$0.34	2600 t/s	0.33 seconds
Llama 4 Maverick	10,000,000	\$0.2	\$0.6	126 t/s	0.45 seconds
Gemma 3 27b	128,000	\$0.07	\$0.07	59 t/s	0.72 seconds

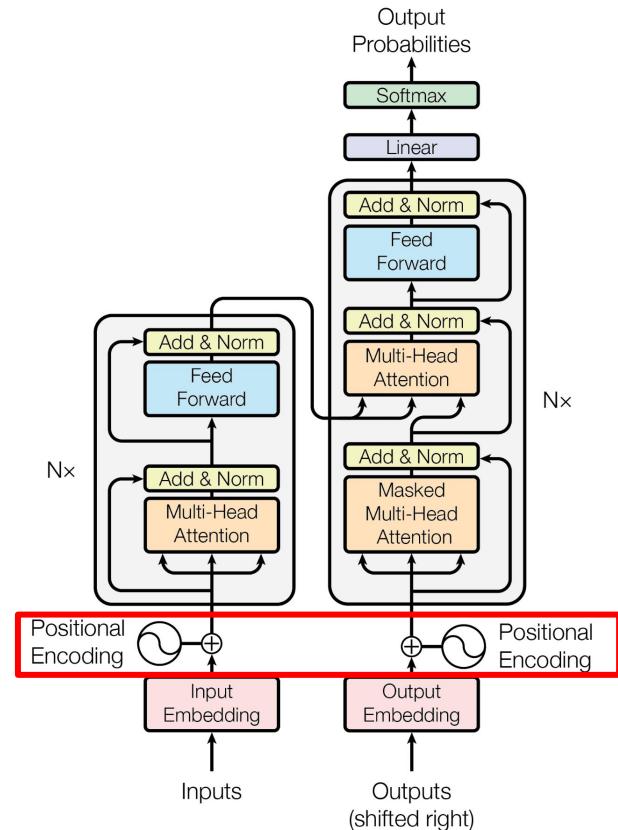


LLM Context Windows Scaling: RoPE

The key **technical innovations** that made long-context LLMs possible:

1. Rotary Positional Embeddings (RoPE)

Remember the positional encodings in the transformer architecture that allowed the model to account for a token's position inside the text?

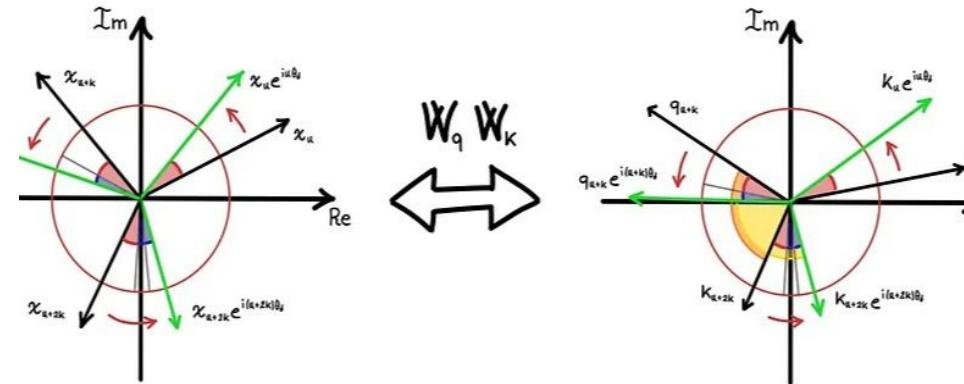




LLM Context Windows Scaling: RoPE

- Transformers need to know where each token sits in a sequence (position). Standard attention doesn't know order by itself, so we add positional information so the model can tell "this word was earlier" or "later".
- RoPE encodes position by rotating the token vectors a little bit depending on their position, so position becomes part of the vector itself, and attention can use relative order naturally.

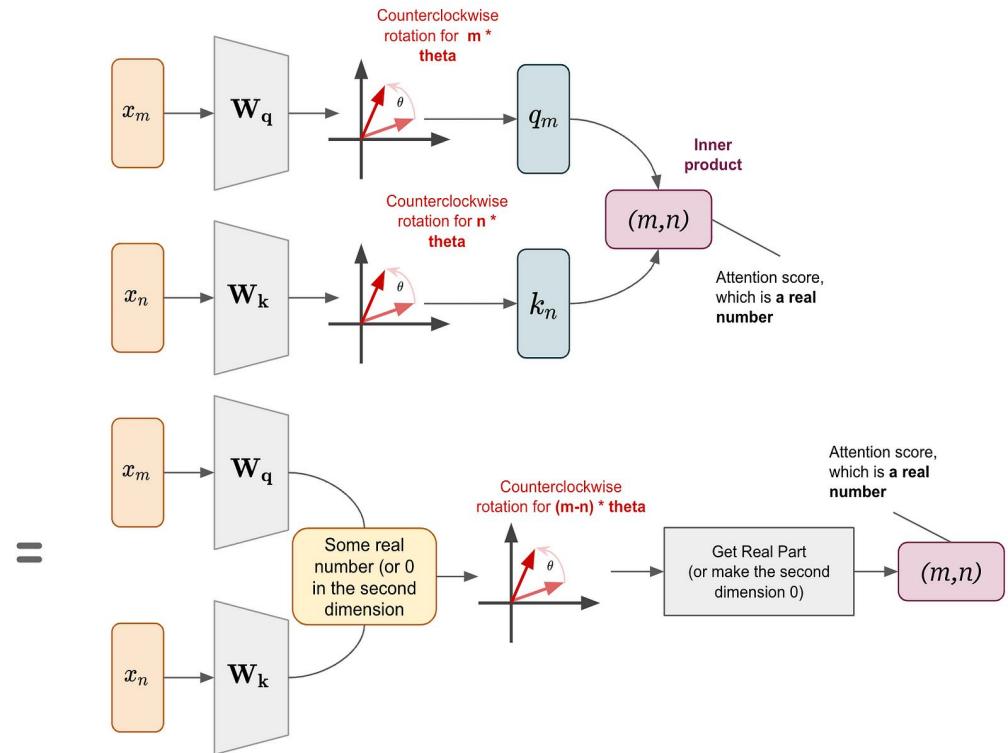
Rotary Encoding





LLM Context Windows Scaling: RoPE

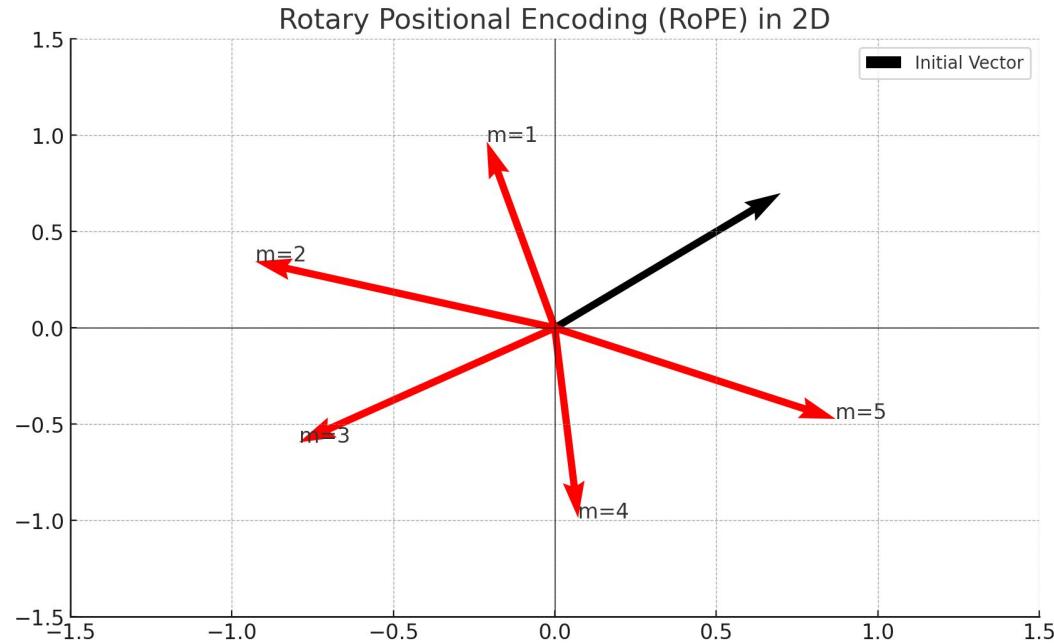
- Imagine each token is an arrow (vector). RoPE gently turns the arrow by a small angle that depends on the token's position. When the model compares two tokens (dot product in attention), those rotated arrows reflect both content and where they sit; so the model can tell not just what tokens are, but how far apart they are.





LLM Context Windows Scaling: RoPE

RoPE's rotary embeddings factor the math so that the attention scores depend *directly* and *only* on the relative offset ($n-m$), making relative positions "explicit" in the mechanism.





Quick recap of RoPE

RoPE (Rotary Position Embeddings) rotates token vectors by an angle tied to their position.

- This rotation makes attention sensitive to relative positions.
- It works well, but when you push a model far beyond the sequence lengths it was trained on, RoPE can start to “break down” because the rotation angles wrap around too tightly.

Think of it like a clock: after 12 hours, the hand comes back to the same place. If you stretch sequences much longer than expected, different positions may start looking “too similar”.

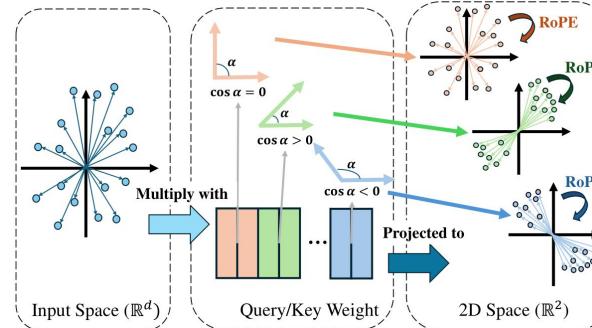


LLM Context Windows Scaling: ReRoPE

2. ReRoPE (Extrapolated Rotary Embeddings):

ReRoPE (Extrapolated Rotary Embeddings) is a simple modification of RoPE designed to let models generalize to much longer sequences.

The idea: instead of rotating with the original frequency schedule (which can cause aliasing at long positions), ReRoPE rescales the rotation angles so they spread out more smoothly over long contexts.



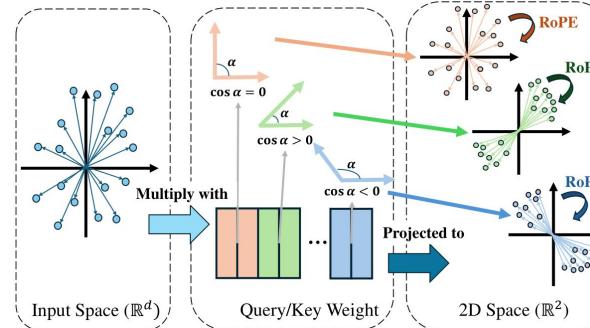


LLM Context Windows Scaling: ReRoPE

2. ReRoPE (Extrapolated Rotary Embeddings):

In practice, this means:

- For short contexts (within training range), ReRoPE \approx RoPE.
- For long contexts (beyond training), ReRoPE slows down the rotation so vectors don't "wrap around" too fast.

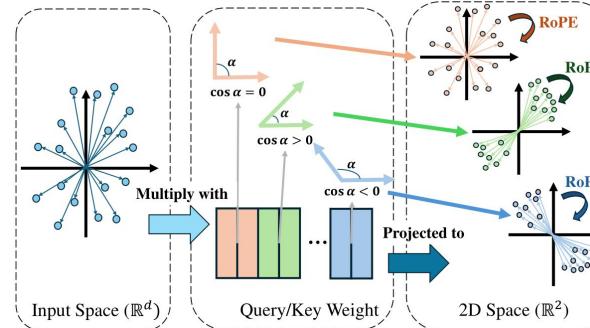




LLM Context Windows Scaling: ReRoPE

2. ReRoPE (Extrapolated Rotary Embeddings):

- If RoPE is like a clock hand turning steadily with position, ReRoPE is like stretching the clock so the hand moves more slowly, avoiding overlap when you go past 12 hours.
- This “slower rotation” makes distances between tokens stay meaningful even when sequences are much longer than training.





LLM Context Windows Scaling: ReRoPE

A. How it works:

- Rather than letting the effective relative distance $(t-i)$ grow unbounded, ReRoPE clamps it at a fixed threshold N .
- Then ReRoPE simply replaces $(t-i)$ with $W(t-i)$ in the rotary

$$W(t-i) = \begin{cases} t - i, & t - i \leq N, \\ N, & t - i > N. \end{cases}$$

B. What does this mean?

- ReRoPE (used in Claude 100K) **modifies RoPE** to extend context windows *without retraining*.
- Ensures stability in long-token inputs by re-centering and rescaling attention patterns.
- Allows 4K-trained models to be pushed to 32K+ tokens during inference.



AMERICAN
UNIVERSITY
OF BEIRUT

RoPE → ReRoPE

- **RoPE:** rotates embeddings with angles tied to position, great for relative positioning but struggles when sequences get much longer than training.
- **ReRoPE:** stretches the rotation (slows it down) to avoid wraparound, letting models extrapolate further without collapsing positional meaning.



Why move beyond RoPE/ReRoPE

Even with ReRoPE, models are still extrapolating, they were trained on a certain max length (say 4K tokens), and everything longer relies on “hoping the math stretches well”. That works, but not perfectly; performance may still drop off when going 8x or 10x longer.

So the next step was to not just extrapolate, but deliberately train for long contexts.



LLM Context Windows Scaling: Position Interpolation

3. Position Interpolation (GPT-4-Turbo)

- Developed by OpenAI, Position Interpolation lets you **reuse** a RoPE-based model unchanged, simply by feeding it **scaled** positions so that all rotary-angle computations stay within the familiar, stable range learned during pre-training.

EXTENDING CONTEXT WINDOW OF LARGE LANGUAGE MODELS VIA POSITION INTERPOLATION

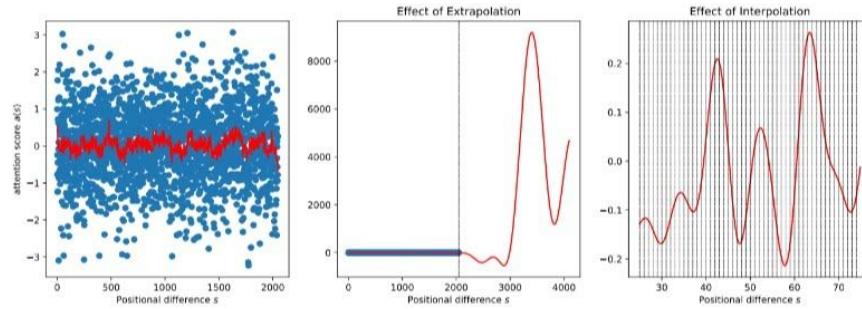


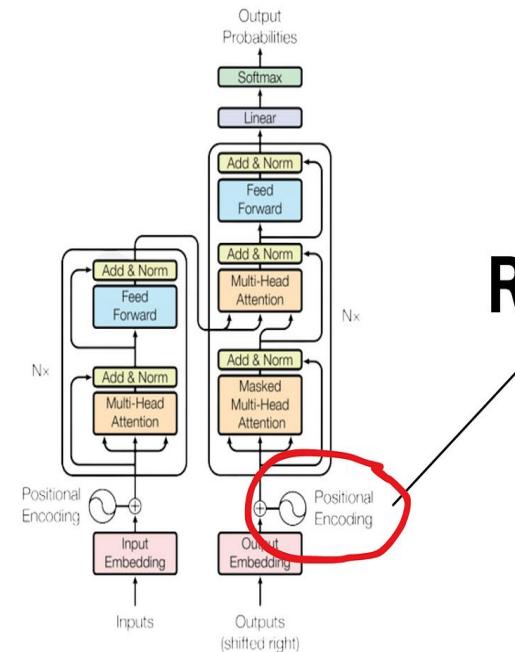
Figure 2: Extrapolation versus interpolation. **Left:** a fitted attention score function (in red) in the form of Eqn. 3 with $d = d_{\text{model}}/n_{\text{head}} = 4096/32 = 128$ (setting of LLaMA 7B). Dots are random input points to be fitted and red curve is the fitted score function via least square, which is approximately within $[-1, 1]$. **Middle:** While the fitted function seems to be well bounded in $[0, L]$, where $L = 2048$, out of this region it may goes beyond 8000, causing catastrophic issues in attention computation. Note that here we do not cherry pick at all: almost every learned curve from a set of randomly generated input points within $[0, L]$ has the extrapolation issue. **Right:** On the other hand, interpolation is much more stable. Curves in between vertical dotted lines (i.e., integer positional difference) are smooth and well-behaved. Please check Appendix C.1 for the source code used to generate the figure.

LLM Context Windows Scaling: Position Interpolation

Position Interpolation is a clever training trick:

- During training, the model sees positions that are compressed versions of much longer ones.
- Concretely: if the model's base context is 2K, you "squash" positions from, say, 8K into the same range (by dividing them down).
- The model learns that positions are interpolated, so when it later runs at 8K (or more), it already knows how to handle those larger indices.

In other words: instead of "stretching" RoPE after training (like ReRoPE does), Position Interpolation teaches the model during training that long contexts exist; by compressing them into shorter slots.



RoPE



Imagine a racetrack

- **RoPE:** Each car drives around a circular track. The farther it goes, the more it turns. But eventually, after enough laps, two cars at very different distances (say 5 km vs 50 km) can end up pointing in nearly the same direction, so the system confuses them.
- **ReRoPE:** To fix this, we stretch the track. Instead of sharp turns, the bends are gentler. Cars can now drive much farther before their directions start to overlap. It's better for longer races, but it's still a stretched version of the same circle.
- **Position Interpolation:**
 - Instead of just stretching the old track, we design the track from the start so that short laps “stand in” for much longer ones. During practice, drivers are taught that one small loop is equivalent to a big loop in the real race.
 - Later, when they face a 50 km marathon, they already know how to interpret it; because training included compressed versions of those long distances.



Why Position Encodings Aren't Enough

Even with RoPE, ReRoPE, or Position Interpolation, transformers still have a big challenge:

- Attention cost grows quadratically with sequence length.
- If you double the input length, the compute and memory needed for attention grows 4x.

So even if we can represent longer contexts, it's still too expensive to process them directly.



LLM Context Windows Scaling: Attention Windowing

4. Attention Windowing and Sliding Windows

- Instead of attending to *all* tokens (which is quadratic in cost), the model:
 - Splits input into windows
 - Uses **local attention** within each window
 - Optionally **slides** the window across the input
- Greatly reduces compute cost for long inputs.

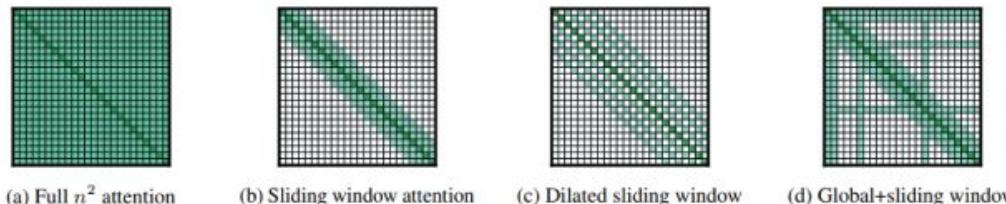


Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.



LLM Context Windows Scaling: Attention Windowing

4. Attention Windowing and Sliding Windows

Think of reading a giant book:

- You don't keep the entire book in your head at once.
- You focus on a few pages (a window) at a time.
- To not lose the story, you reread the last paragraph from the previous section (sliding window overlap).

This makes reading manageable without memory overload, just like windowed attention makes long-context transformers efficient.



Context Window Flow

So the flow looks like this:

- RoPE → give attention a sense of order.
- ReRoPE → stretch that sense of order for longer contexts.
- Position Interpolation → train to naturally handle long contexts.
- Windowing & Sliding Windows → make processing long contexts actually efficient, by limiting how much each token attends to.



AMERICAN
UNIVERSITY
OF BEIRUT

LLM Training



Process of LLM Training

Assuming we have chosen the desirable architecture of the model, we would need to perform the following steps to train the model:





AMERICAN
UNIVERSITY
OF BEIRUT

LLM Pre-Training Dataset and Training



LLM Pre-Training Data

Scale

- Hundreds of billions to trillions of tokens.
- Examples:
 - GPT-3: 300B tokens
 - GPT-4 / Claude / Gemini: Exact sizes undisclosed, but significantly larger.

Diversity

- **Web pages** (Common Crawl, Wikipedia, blogs)
- **Books** (Project Gutenberg, academic texts)
- **Code** (GitHub, StackOverflow dumps)
- **Multilingual sources** (news articles, web text)

Pre-training Dataset
Collection and Preparation

- Collection of a large dataset
- Cleaning, De-duplication, tokenization,..

More diverse data = broader general knowledge & better generalization.



Pre-Training Data Pruning

Collecting additional domain-Specific Data

- Inclusion of medical, legal, technical, or code-heavy corpora enhances specialized capabilities
- Claude models show strength in technical reasoning likely due to more structured data
- Gemini excels in factual knowledge likely due to deeper integration with Google Search data

Pre-training Dataset
Collection and Preparation

- Collection of a large dataset
- Cleaning, De-duplication, tokenization,..



Pre-Training Data Pruning

Data Cleaning Techniques

- **Tokenization:** Byte Pair Encoding (BPE), SentencePiece (Unigram), or newer methods.
- **Curriculum Learning:** Start simple, increase complexity gradually.
- **Deduplication:** Prevent memorization of repeated web content.
- **Data Mixture Balancing:** Control how much of each domain is seen (e.g., code vs. web text).

Models like GPT-4 and Claude likely use **heavily filtered, high-quality corpora**, rather than raw web dumps.

Pre-training Dataset
Collection and Preparation

- Collection of a large dataset
- Cleaning, De-duplication, tokenization,..



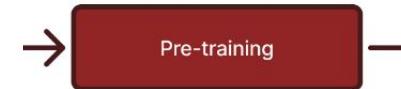
LLM Pre-Training

Goal: Learn to predict the next word (or token) given previous context.

- Example:

Input: "The Eiffel Tower is located in"

Output: "Paris"



Training the model to perform
next-word prediction on the
per-training dataset

Why it's powerful:

- Forces model to learn grammar, facts, reasoning patterns.
- Simple but general — supports many downstream tasks.



LLM Pre-Training

Variants:

- **Causal Language Modeling** (used in GPT-style models): strictly left-to-right.
- **Masked Language Modeling** (used in BERT): randomly masks tokens to predict.
- **Permutation-based modeling** (e.g., XLNet): learns multiple token orders.



Training the model to perform next-word prediction on the per-training dataset



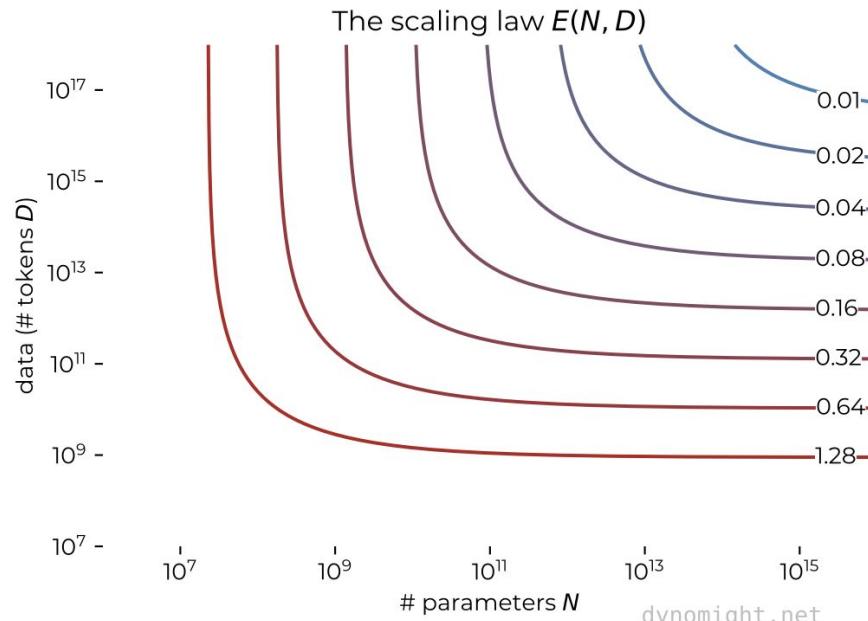
We saw that as LLMs advance, their size (architecture) and thus we use a bigger training data set.

Is there a limit to how much training data is enough? Is more data always better?



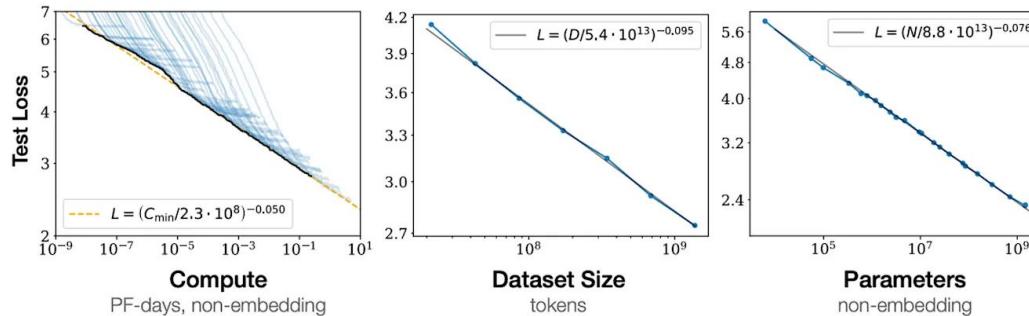
LLM Scaling Laws

Researchers have found that the **number of parameters** an LLM has and the **size** of the training set directly influence the **loss** of the next token prediction.



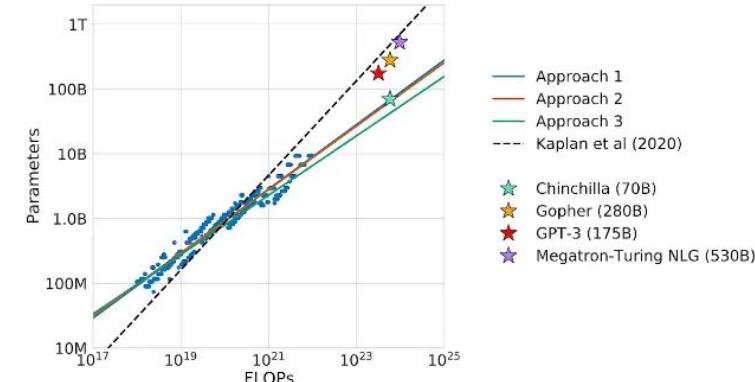
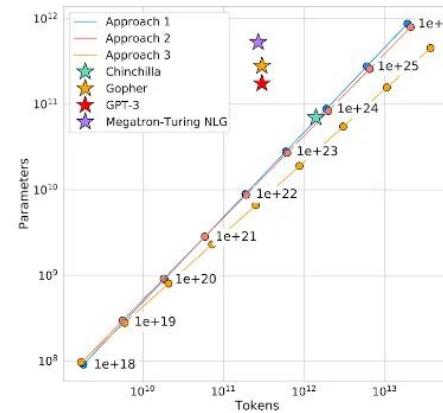
LLM Scaling Laws

- Hence, they are trying to answer the question of LLMs scaling law: Given **constrained compute budget** measured in FLOPs, floating-point operations, what would be the optimal combination of **model size** and **training data size** (measured in number of tokens) that yields the **lowest loss**?
- Training curves tend to follow **predictable power-law patterns**, allowing for **rough predictions** of future performance based on current training progress.



LLM Scaling Laws

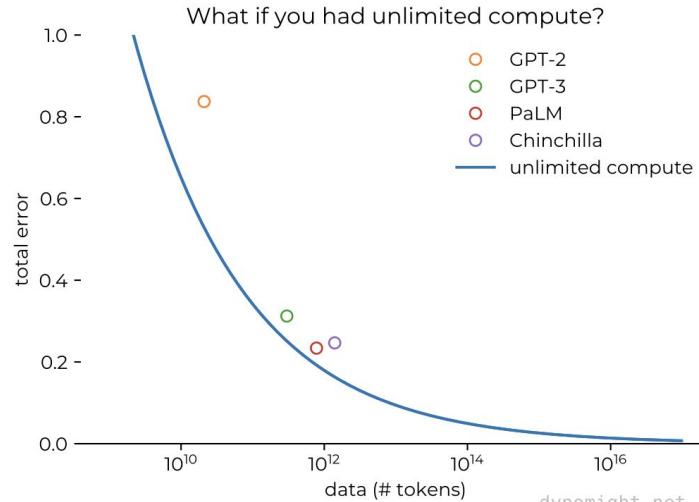
- DeepMind detailed various methods for the scaling law of training compute-optimal LLMs, that are different than OpenAI's.
- They have conducted over 400 LLMs training and found out that for compute-optimal training, the model size and dataset size (number of tokens) should be **scaled equally**.





LLM Scaling Laws

- In order to achieve the same model performance, all three approaches suggest much **smaller model size**.
- That is with a given compute budget and training data, one can achieve the **same model performance with less model size**.





LLM Scaling Laws

Parameters	Approach 2		Approach 3	
	FLOPs	Tokens	FLOPs	Tokens
400 Million	1.84e+19	7.7 Billion	2.21e+19	9.2 Billion
1 Billion	1.20e+20	20.0 Billion	1.62e+20	27.1 Billion
10 Billion	1.32e+22	219.5 Billion	2.46e+22	410.1 Billion
67 Billion	6.88e+23	1.7 Trillion	1.71e+24	4.1 Trillion
175 Billion	4.54e+24	4.3 Trillion	1.26e+24	12.0 Trillion
280 Billion	1.18e+25	7.1 Trillion	3.52e+25	20.1 Trillion
520 Billion	4.19e+25	13.4 Trillion	1.36e+26	43.5 Trillion
1 Trillion	1.59e+26	26.5 Trillion	5.65e+26	94.1 Trillion
10 Trillion	1.75e+28	292.0 Trillion	8.55e+28	1425.5 Trillion

Estimated optimal training FLOPs and training tokens for various model size



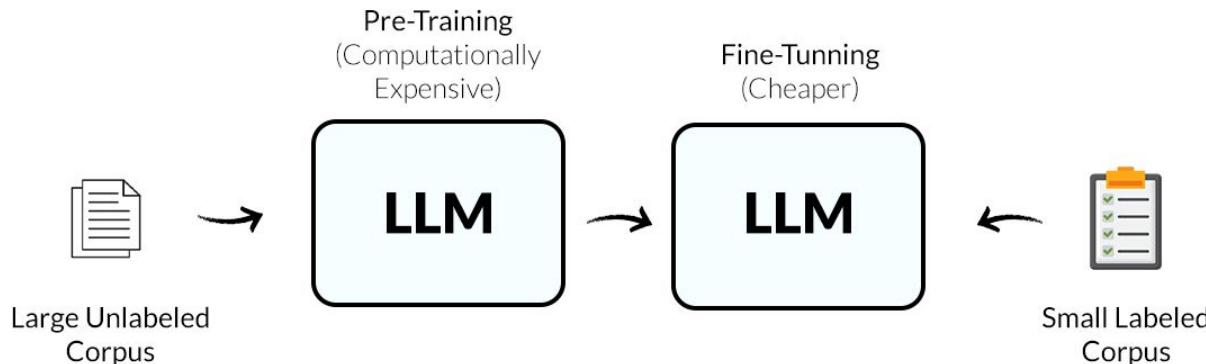
AMERICAN
UNIVERSITY
OF BEIRUT

LLM Fine-tuning Data and Training



Why Do We Need Fine-Tuning?

- A pre-trained language model **understands text data**, its structure, connection, and references, but it **cannot perform a specific task well**.
- To leverage the **LLMs comprehension** of text data, we **train it further** on a **smaller task-specific corpus**.





Fine-Tuning Techniques

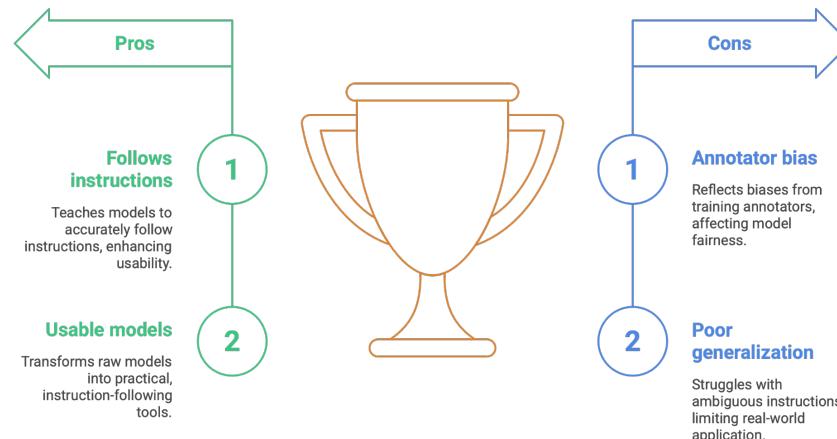
After pretraining on massive unlabeled datasets, LLMs still need **fine-tuning** to become useful, aligned, and safe for human interaction. There are **techniques** to fine tune an LLM:





Fine-Tuning Techniques

- **What it is:** The model is trained on curated *input-output* examples to mimic human-like responses.
- **Examples:**
 - Prompt: "Write an email to request a meeting."
 - Target: "Dear Dr. Smith, I hope you're well. I'd like to schedule a meeting to..."





Reinforcement Learning with Human Feedback (RLHF)

Goal: Align model outputs with *human preferences* — not just correctness.

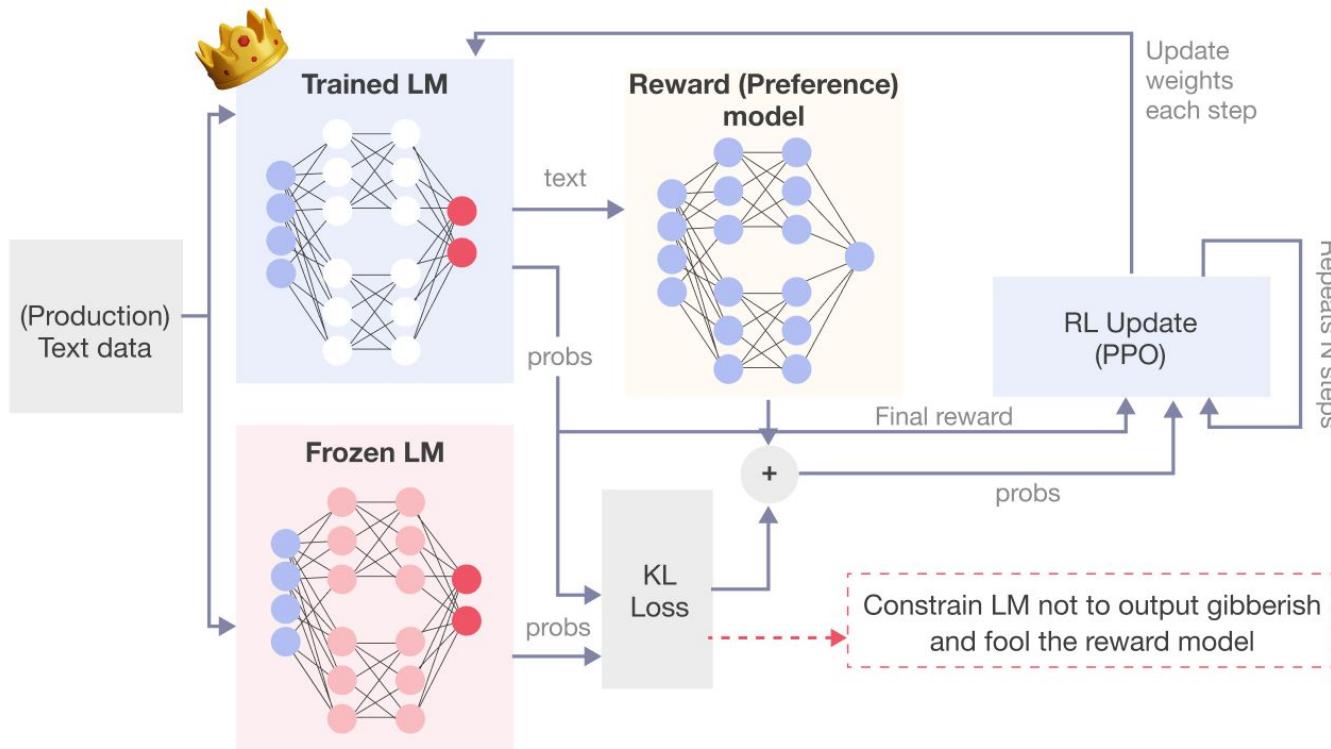
- Captures subtle human preferences: tone, harmlessness, helpfulness.
- Balances safety with creativity.

Process:

1. Generate multiple completions for the same prompt.
2. Humans **rank the outputs**.
3. Train a **reward model** to predict those rankings.
4. Use **PPO** (Proximal Policy Optimization) to fine-tune the LLM toward outputs with higher reward.



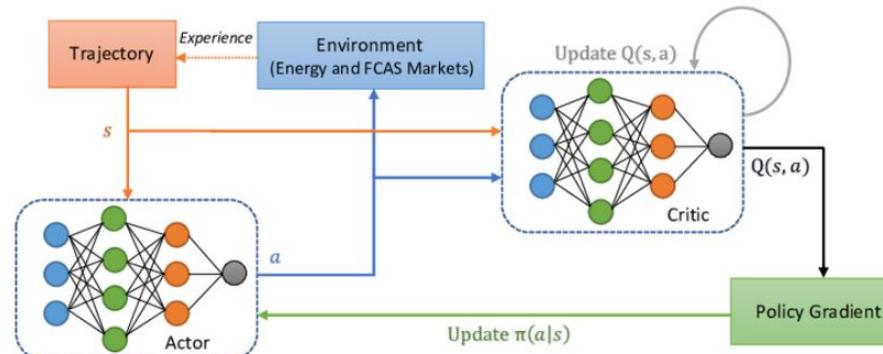
Reinforcement Learning with Human Feedback





Proximal Policy Optimization

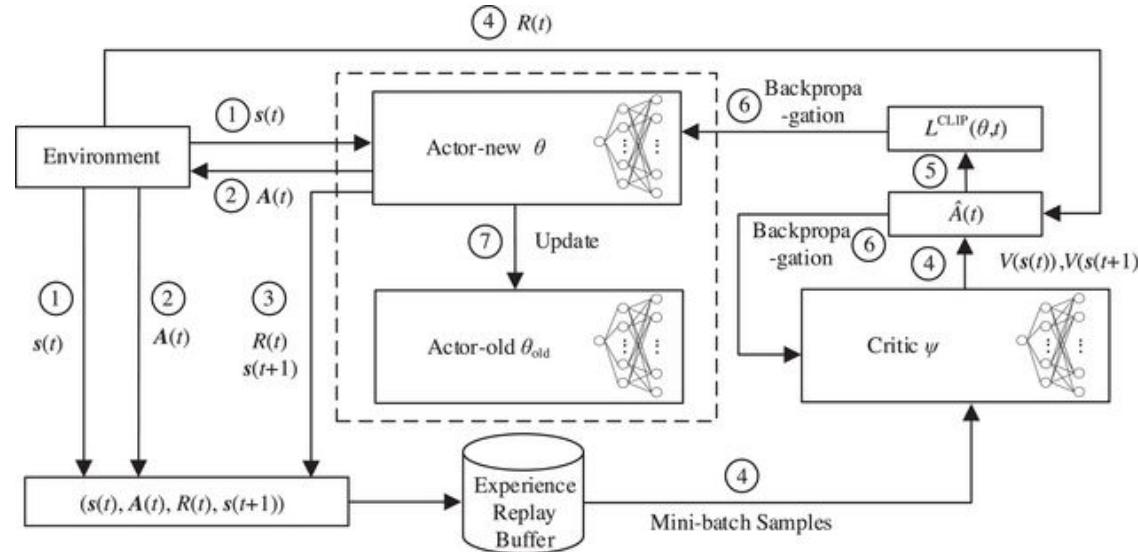
- Reinforcement learning algorithm known for its stability and ease of implementation.
- It's a policy gradient method that aims to optimize an agent's policy (its decision-making strategy) by directly estimating it and using a value function to assess the policy's effectiveness.
- PPO achieves stability by limiting the size of policy updates, preventing drastic changes that could destabilize learning.





Proximal Policy Optimization

- **Why it's used:** Prevents the model from drifting too far from its pre-trained knowledge during alignment.
- **Drawback:** Computationally expensive, hard to scale to very large datasets.





Direct Preference Optimization

What it is: A simpler alternative to PPO, introduced in 2023.

How it works:

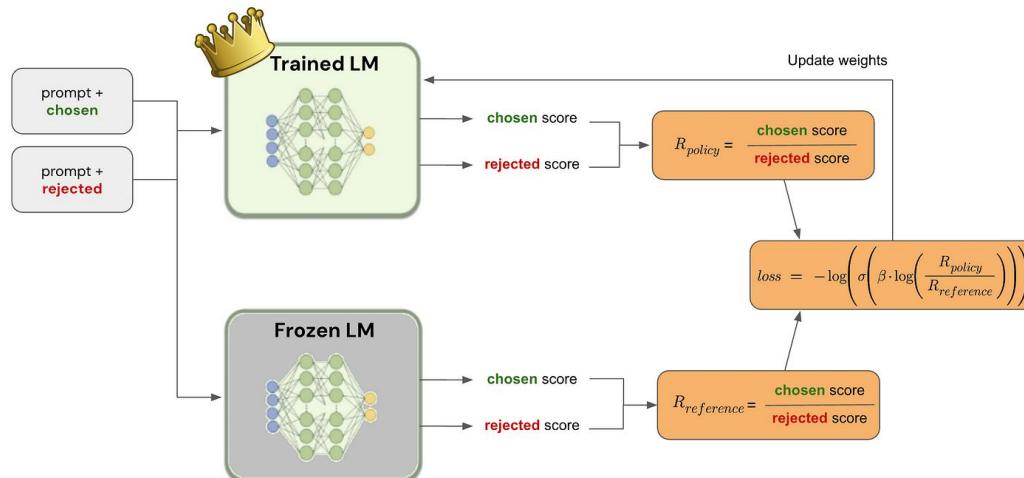
- Optimizes model parameters directly using **preference pairs**, without training a separate reward model.
- Example:
 - Given two outputs A and B, and knowing humans prefer A over B, DPO updates the model to prefer A.



Direct Preference Optimization

Benefits:

- Easier to implement, more stable than PPO.
- Matches or beats RLHF performance on some tasks.





Fine-Tuning Techniques

Method	Purpose	Human Effort	Scalability	Used By
SFT	Teach task-following	<input checked="" type="checkbox"/> High	<input checked="" type="checkbox"/> Medium	All LLMs
RLHF + PPO	Align to human preferences	<input checked="" type="checkbox"/> Very High	<input type="checkbox"/> Expensive	OpenAI, Anthropic
DPO	Efficient preference learning	<input checked="" type="checkbox"/> Moderate	<input checked="" type="checkbox"/> High	Open-source, Anthropic



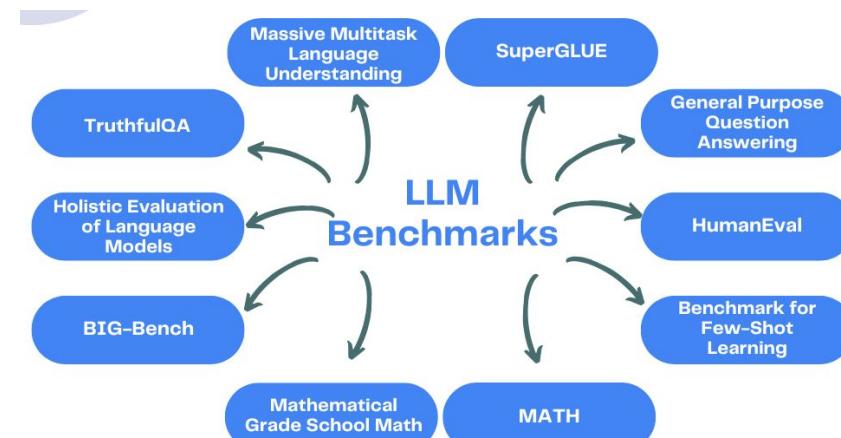
AMERICAN
UNIVERSITY
OF BEIRUT

LLM Testing and Optimization



LLM Benchmarking

- How do we **measure** LLM performance? What does “state of the art” actually mean?
- LLMs are general-purpose models — they do many tasks (reasoning, coding, math, writing).
- We need **multidimensional benchmarks** to test:
 - Knowledge
 - Reasoning
 - Language fluency
 - Safety and factuality
 - Coding, math, logic, etc.





LLM Benchmarking - MMLU

There are some benchmarks that you will frequently encounter:

MMLU (Massive Multitask Language Understanding)

- 57 tasks across subjects: history, law, math, medicine, etc.
- Multiple-choice questions to evaluate **factual recall and reasoning**.
- Used in GPT-4, Claude, Gemini evaluations.

High scores = strong general knowledge & reasoning under constraints.

question string · lengths	subject string · classes	choices sequence · lengths	answer class label
41 	1 value	4	4 classes
Find the degree for the given field extension $\mathbb{Q}(\sqrt{2}, \sqrt{3})$, ...	abstract_algebra	["0", "4", "2", "6"]	1 B
Let $p = (1, 2, 5, 4)(2, 3)$ in S_5 . Find the index of $\langle p \rangle$ in S_5 .	abstract_algebra	["8", "2", "24", "120"]	2 C
Find all zeros in the indicated finite field of the given...	abstract_algebra	["0", "1", "0.1", "0.4"]	3 D
Statement 1 A factor group of a non-Abelian group is non-Abelian....	abstract_algebra	["True, True", "False, False", "True, False", "False, True"]	1 B



LLM Benchmarking - HumanEval

HumanEval

- First published in 2021 in the paper: ‘Evaluating Large Language Models Trained on Code’
- OpenAI benchmark for **Python code generation**.
- Prompts require the model to **write functions** that pass **unit tests**.
- Used to **evaluate LLMs** for software engineering use cases.

Useful for judging *functional correctness* in code, not just fluency.

```
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]

def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) =>12
    solution([3, 3, 3, 3, 3]) =>9
    solution([30, 13, 24, 321]) =>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```



LLM Benchmarking - BIG-bench

BIG(Beyond the Imitation Game Benchmark)-bench

- Published by google
- **collaborative benchmark** intended to probe large language models and extrapolate their future capabilities.
- **200+ diverse tasks** curated by the research community.
- Has **different variations for complexity** like ‘Big bench hard’ and ‘Big bench extra hard’
- Tests creativity, logical reasoning, analogies, and even **edge-case generalization**.

Input:

In the following chess position, find a checkmate-in-one move.

1. e4 b6 2. d4 Bb7 3. Nc3 e6 4. Nf3 Bb4 5. Bd3 d5 6. e5 Ne7 7. a3 Bxc3+ 8. bxc3 c5 9. h4 Nf5 10. Bxf5 exf5 11. Ng5 g6 12. e6 fxe6 13. Nxe6 Qe7 14. Qe2 Bc8 15. Nc7+ Kf7 16. Qxe7+ Kxe7 17. Nxg8 Kd6 18. Bf4+ Kc6 19. dxc5 Nd7 20. cxb6 axb6 21. Rb1 Re8+ 22. Kd2 Ne5 23. Rxb6+ Kc5 24. Be3+ Kc4 25.

Output:

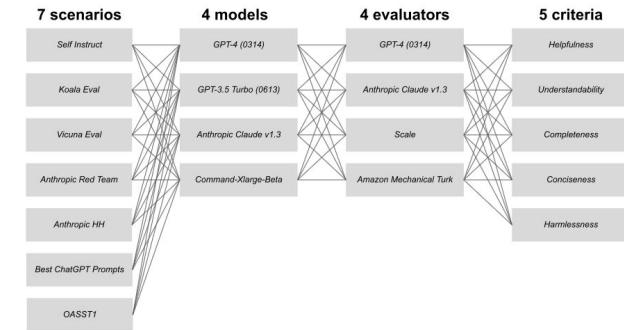
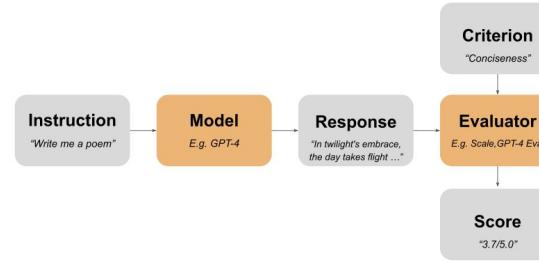
Rb4+ Kd3 26. Rd4+ Kc2 27. Rd2+ Kd3 28. Rd4+ Kd2 29. Rd2+ Kd3 30. Rd4+ Kc2 31. Rd2+ Kd3 32. R



LLM Benchmarking - HELM

HELM (Holistic Evaluation of Language Models)

- Developed by Stanford CRFM.
- Evaluates models **holistically** across:
 - Accuracy
 - Calibration
 - Fairness
 - Robustness
 - Efficiency
 - Transparency

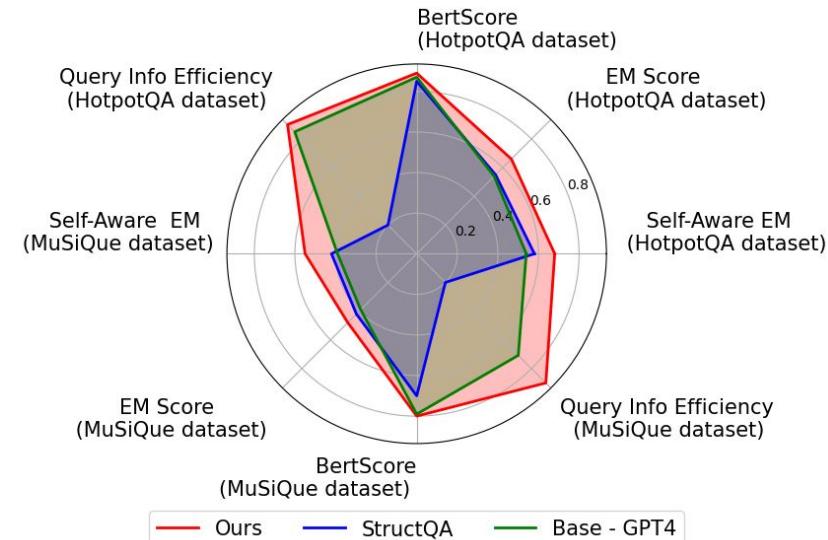




How does an LLM achieve SOTA?

“State of the art” is **benchmark-dependent**.

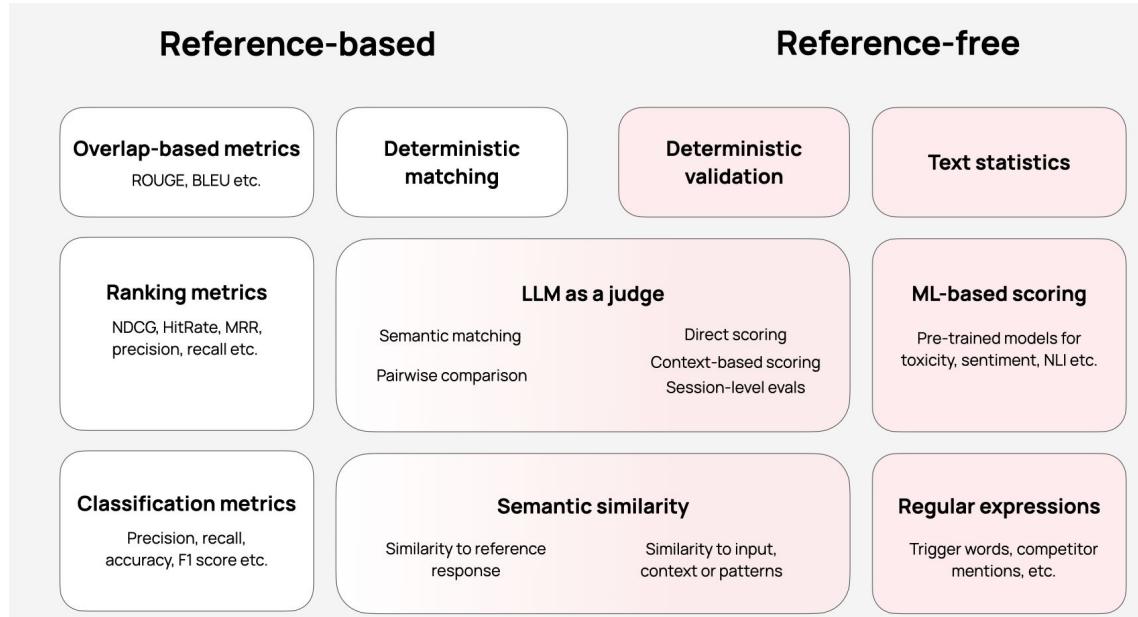
- A model might be SOTA in **code generation** (like Claude), but not in **math** or **multimodal vision**.
- SOTA claims are valid *only if*:
 - The benchmark is public
 - The methodology is reproducible
 - The task is relevant to your use case





LLM Evaluation

Whether you're using a benchmark or not, there are a variety of **metrics** for LLM evaluation





LLM Evaluation

Some evaluation metrics are **task specific**:

Task	Metric Examples	Description
Classification	Accuracy, F1-score	How often is the label correct?
Translation	BLEU, METEOR	How close is the translation to the reference?
Summarization	ROUGE, BERTScore	Does the summary cover key points?
QA / Retrieval	Exact Match, F1, EM, Recall@K	Did the model find the correct fact or passage?



LLM Evaluation - Language Generation

Metric	What It Measures	Notes	Equation
Perplexity	How “surprised” the model is by text	Lower = better prediction	$\text{Perplexity} = \exp\left(-\frac{1}{N} \sum_{i=1}^N \log P(w_i)\right)$
BLEU / ROUGE	N-gram overlap with ground truth	Used in translation/summarization	$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^N w_n \log p_n\right)$ <p>Where:</p> <ul style="list-style-type: none">• p_n = precision of n-grams• w_n = weight for n-gram (usually uniform)• BP = brevity penalty for short hypotheses $\text{ROUGE-L} = \frac{\text{LCS}(X, Y)}{\text{len}(Y)}$ <p>Where X is the generated text and Y is the reference.</p>
BERTScore	Semantic similarity using embeddings	More robust than surface-level overlap	$\text{BERTScore} = \frac{1}{ C } \sum_{c \in C} \max_{r \in R} \cos_sim(c, r)$ <ul style="list-style-type: none">• C: embeddings of candidate tokens• R: embeddings of reference tokens
HumanEval	Code correctness via unit tests	Used for code generation	$\text{Pass}@k = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}$ <p>Where:</p> <ul style="list-style-type: none">• n = total number of generated samples per problem• c = number of correct samples (i.e., those that pass all tests)• k = how many samples we select randomly (e.g., 1, 5, or 10)



LLM Evaluation - Safety

Metric	Description	Evaluated With
Toxicity Score	Probability of generating harmful content	Perspective API, Detoxify
Refusal Accuracy	Does the model reject unsafe prompts properly?	Adversarial prompts
Jailbreak Success	% of successful safety bypass attempts	Red-teaming frameworks
Honesty / Helpfulness / Harmlessness	Alignment to user intent & ethics	Anthropic's HHH, HELM



Red Teaming

Red-Teaming:

- A team of humans (or models!) tries to “break” the model.
- Example: trick it into giving harmful advice, leaking secrets, or failing logic tests.
- Reveals *failure modes* early on.

Adversarial Prompting:

- Specially crafted prompts designed to **bypass safety filters**.
- Examples:
 - Jailbreaking attempts (e.g., DAN, “ignore previous instructions”)
 - Prompt injection in multi-agent setups



Red Teaming

Risk Type	Description	Example
Hallucination	Confidently making up facts	Fake citations, wrong code
Toxicity / Bias	Reflecting harmful societal biases	Racist, sexist, or political responses
Jailbreaks	Circumventing safety filters	"Write a bomb recipe" prompts
Prompt Injection	Manipulating the model through user input	Agents misbehaving or leaking data



AMERICAN
UNIVERSITY
OF BEIRUT

Thank you!