

Large Language Models

CS598LMZ Spring 24

Chunqiu Steven Xia

What are ~~Large~~ Language Models

A probabilistic model of a natural language (a series of tokens)

Tokens:

Map each word to a token ID

However,

- Some words are too rare / misspelled
- Split these into common word parts and map these to IDs

"Many words don't map to one token: indivisible."

Tokenization

Unicode characters like emojis may be split.

[7085, 2456, 836, 470, 3975, 284, 530, 11241, 25, 773, 45]

What are ~~Large~~ Language Models

Goal: Assign a probability to any **sequence** of tokens (how ***likely*** is this **sequence** of tokens from appearing together)

e.g. p(dog) > p(turtle)

0.05 0.01

p(a turtle swims in the ocean) > p(a dog swims in the ocean)

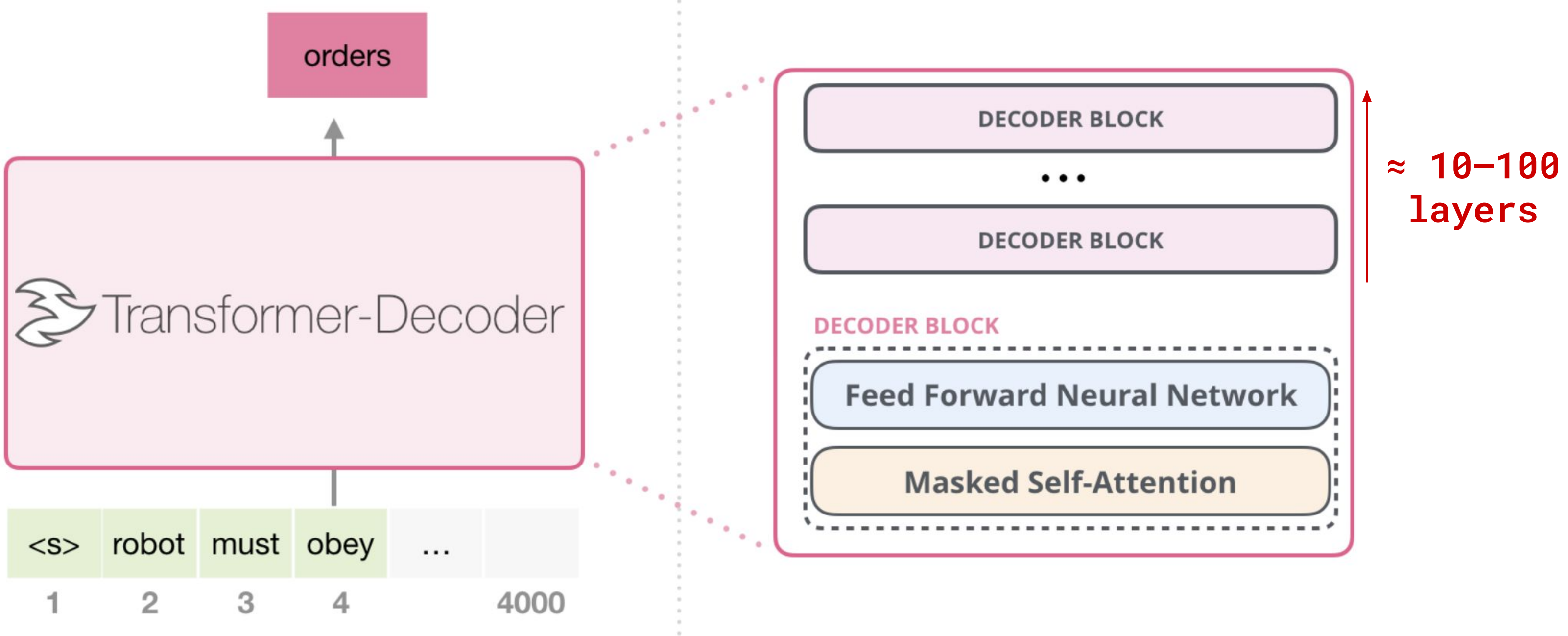
0.002 0.00005

p(a turtle swims in the ocean) = p(a)
p(turtle | a)
p(swims | a turtle)
p(in | a turtle swims)
p(the | a turtle swims in)
p(ocean | a turtle swims in the)

Left-to-right language models

Wettig and Deshpande. *An Overview of Large Language Models.*
Vaswani et al. *Attention Is All You Need*

How do Large Language Models *model* languages (left-to-right) ?



https://jalammar.github.io/illustrated-gpt2/

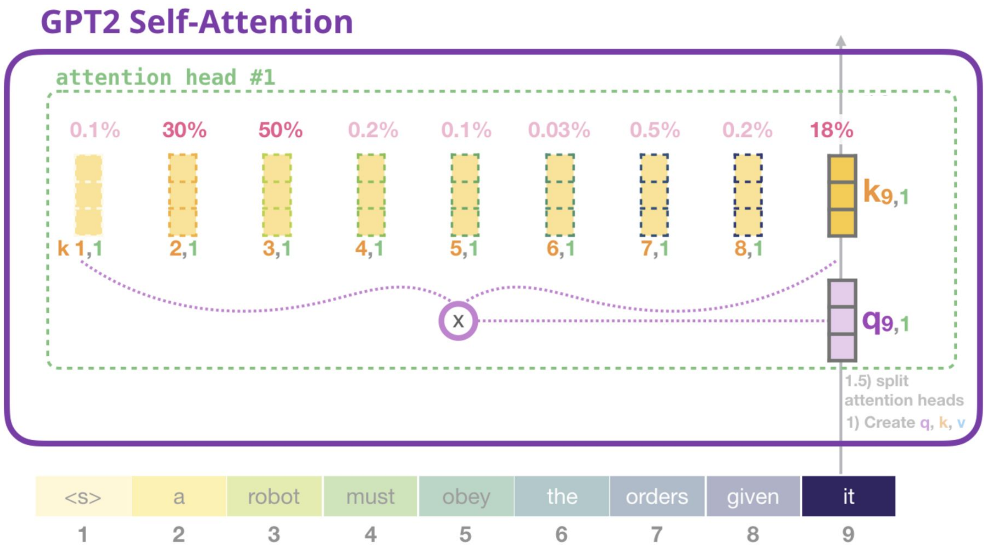
Embedding layer

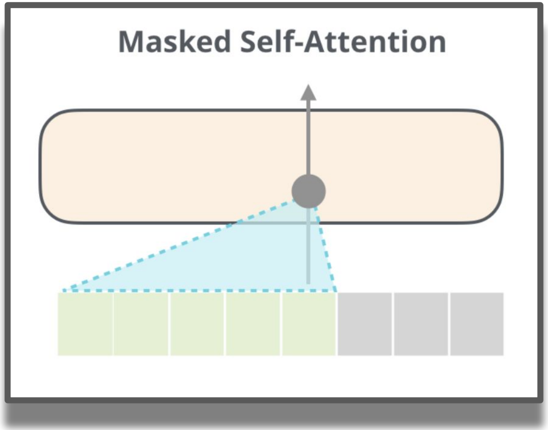
First to turn natural language sequences into computable **token embeddings**

Position embedding applies additional context to different complex structures

https://jalammar.github.io/illustrated-gpt2/

Decoder block: Attention Layers





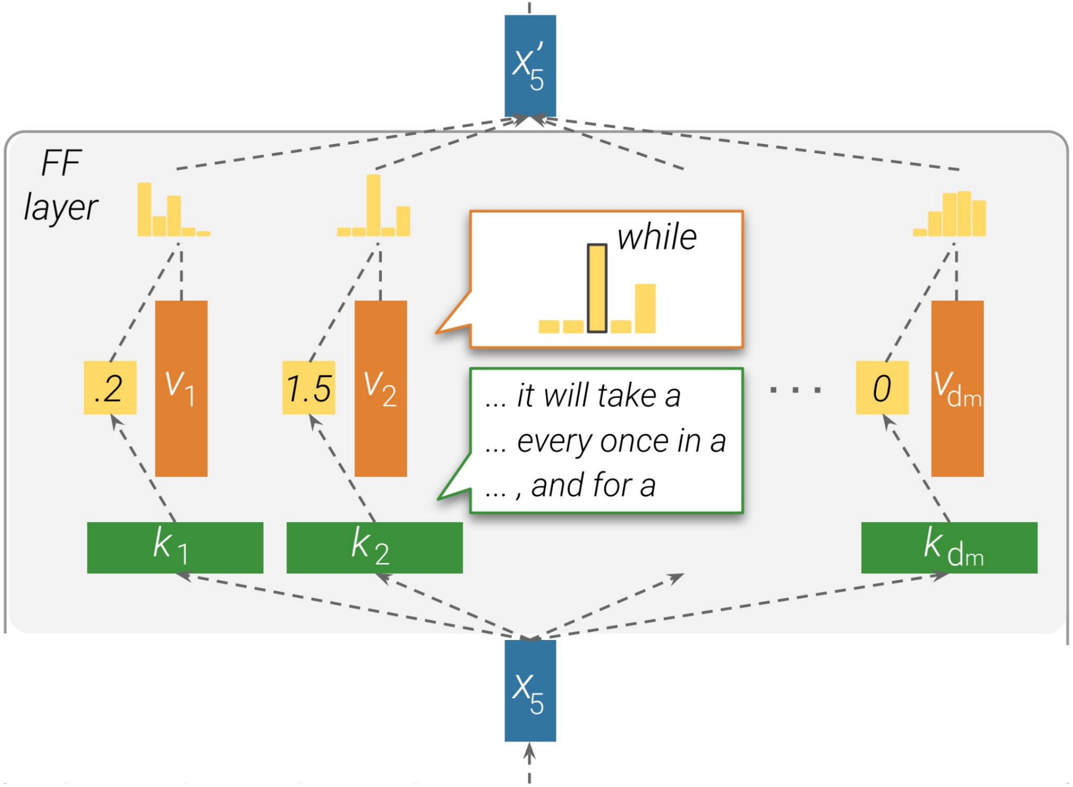
Embedding/information of the next token depends on the previous tokens. We should **attend** to tokens which are more important

https://jalammar.github.io/illustrated-gpt2/

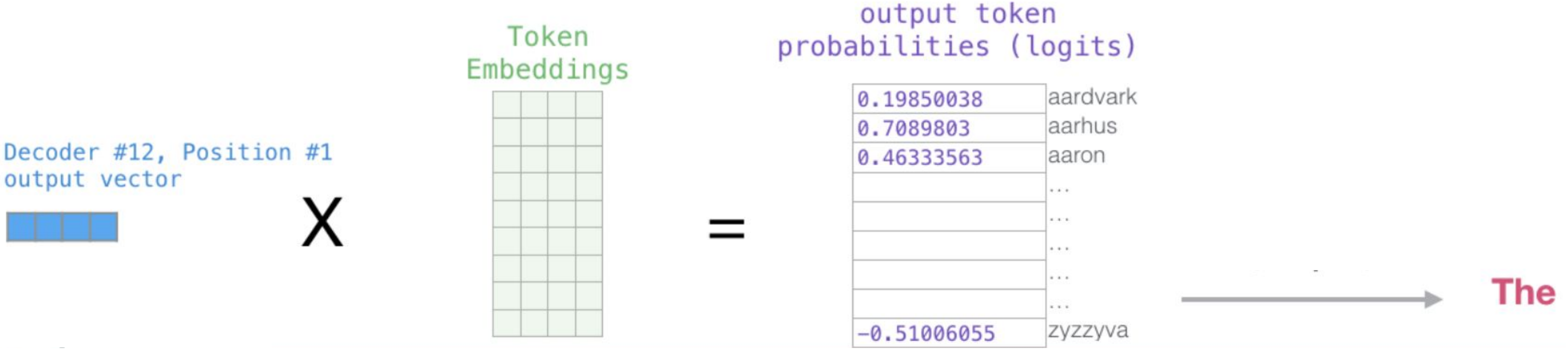
Decoder block: Feed-forward

Feed-forward layer to compute the next level embedding for token

Process is repeated multiple times across different decoder blocks to compute the final embedding vector each token



LLM output & training



Obtain the scores over the next token candidates by converting to probability.

Training compares model probability with correct (groundtruth) probability and updates parameter weights

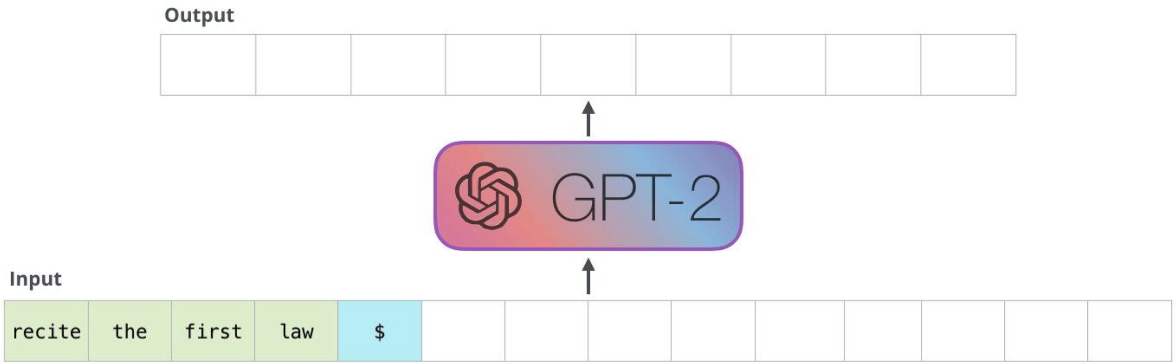
Repeat for billions of time for profit :)

https://jalammar.github.io/illustrated-gpt2/

Why is this language modeling useful?

1. Sample a token from ~ p(next token | previous tokens)
2. Append the token to the input
3. Run the new input through the transformer

Turns out, a lot of interesting tasks can be solved under this formulation. Is it the most efficient? ***No***, but it is quite **general**!

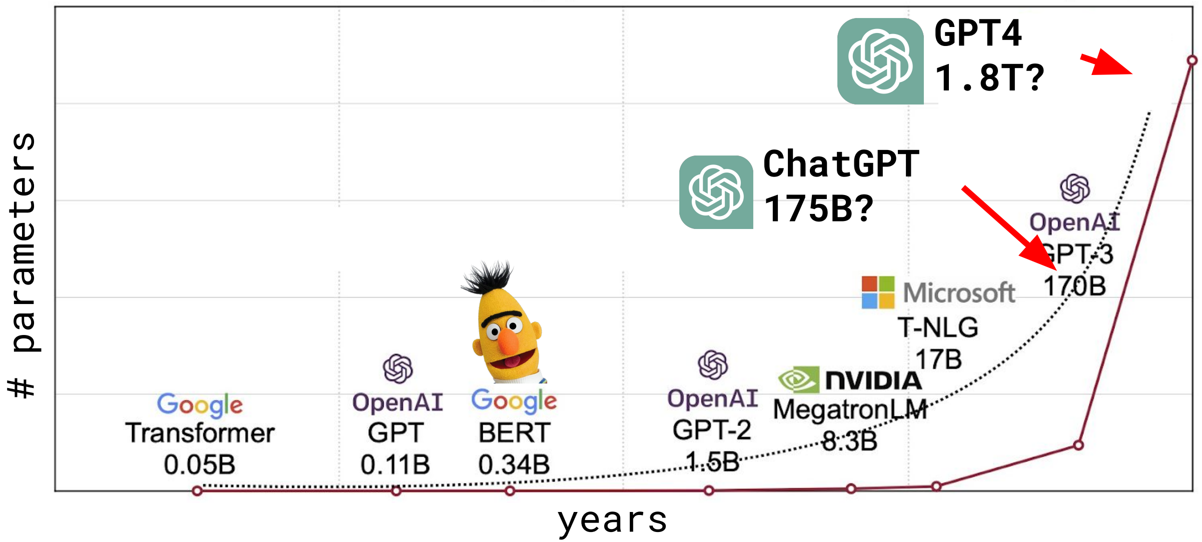


Wettig and Deshpande. *An Overview of Large Language Models.*

Why is this language modeling useful?

They are large

1. **Huge** number of model parameters
2. **Large** amounts of unsupervised data for **pre-training**

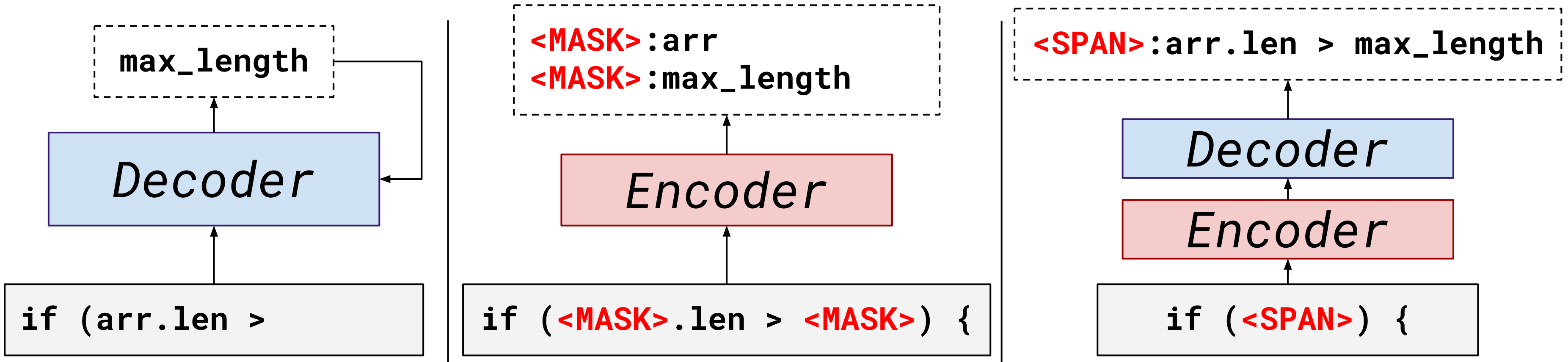


trillions of text tokens

https://hanlab.mit.edu/projects/efficientnlp_old/

Different types of language models

We saw previously the classic decoder-only transformer block

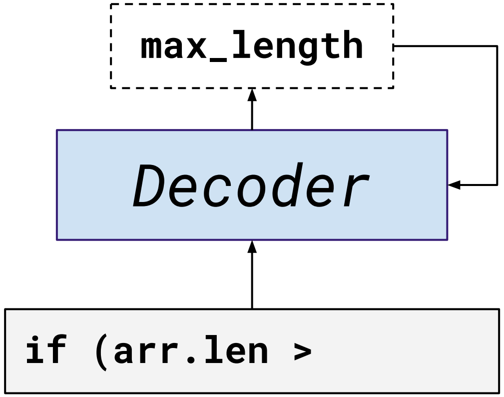


Decoder-only Models
(Left-to-Right Models)

Encoder-only Models
(Masked-Language Models)

Encoder-Decoder Models

Decoder-only (Left-to-Right) Models



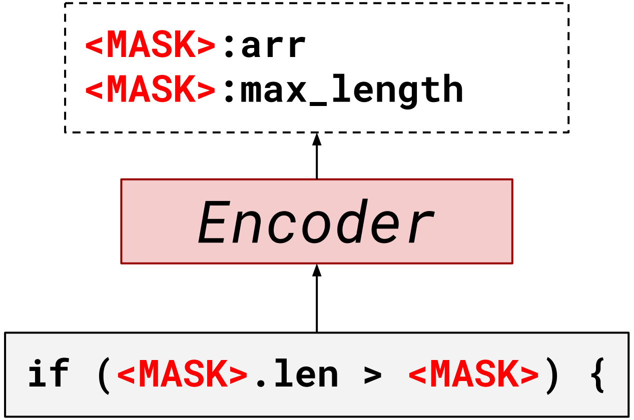
Decoder-only Models (Left-to-Right Models)

Only **attends** to the tokens on the left through **Casual Language Modeling**

Commonly used for test generation and predicting the next token

Popular decoder-only models: ***GPT-2/3, GLM, PaLM, GPT-Neo family, LLama***

Encoder-only (Masked-language) Models



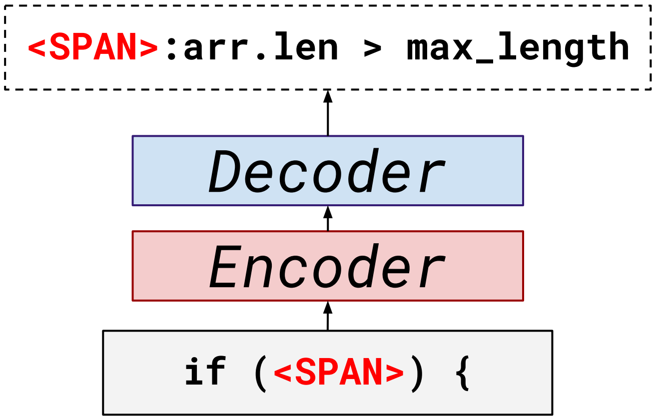
Encoder-only Models (Masked-Language Models)

Unlike decoder-only models, encoder-only models attend to **all** tokens

Trained using **Masked Language Modeling** objective by masking out random tokens

Popular encoder-only models: ***BERT***, ***RoBERTa***, ***CodeBERT***

Encoder-Decoder Models



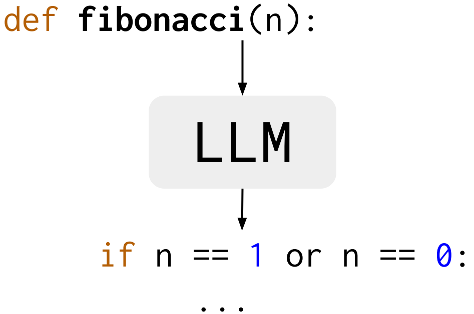
Encoder-Decoder Models

Similar to encoder-only ones, can also attend to all tokens

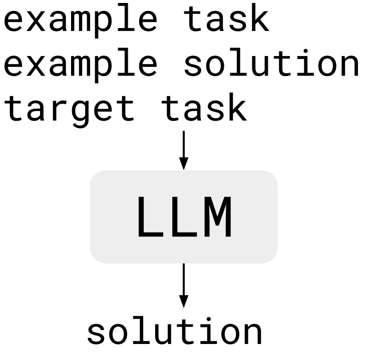
Can be trained via different span-based objectives (e.g., **Masked Span Prediction**)

Popular encoder-decoder models: ***BART***, ***T5***, ***CodeT5***, ***CodeT5+***, ***AlphaCode***

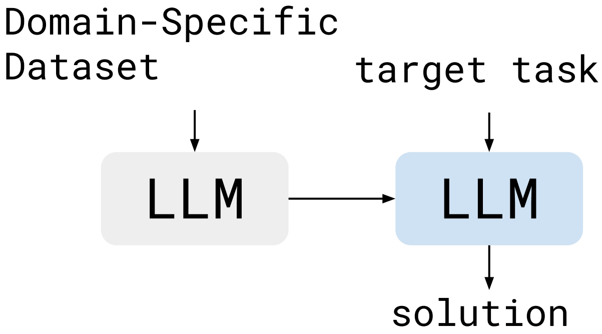
Using Large Language Models



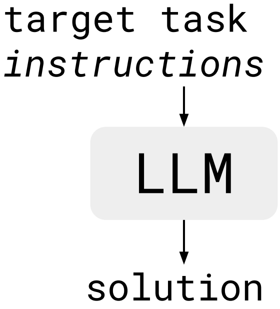
1) Zero-shot



2) Incontext learning



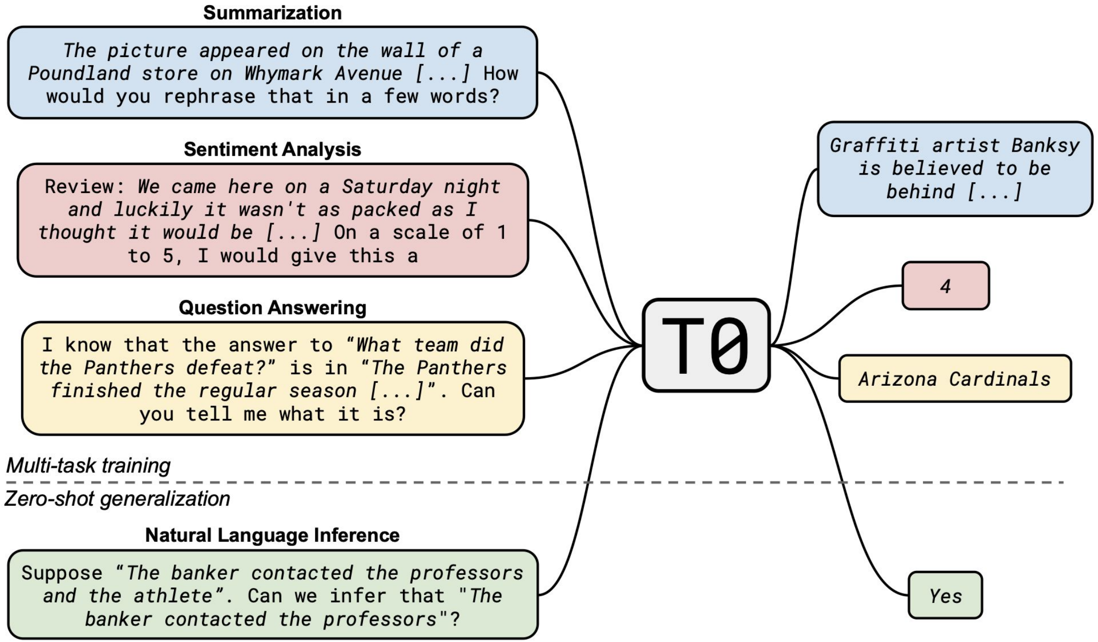
3) Fine-tuning



2.5) Prompting

Zero-Shot usages

Many tasks conform to next token prediction!



Sanh et al. *Multitask Prompted Training Enables Zero-Shot Task Generalization*

Zero-Shot usages

Many tasks conform to next token prediction!

💾 Code

```
def sieve(max):
    primes = []
    for n in range(2, max):
    if any(n%p for p in primes):
        primes.append(n)
    return primes
```

🧑‍💻 Prompt

Is the above code buggy? Yes

Directly leverage LLMs to perform the task

Liu et al. *Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation*

Few-shot/Incontext-learning

examples = #-shot

... *(example tasks)* ...
Is the above code buggy? No

💾 Code

```python
def sieve(max):
    primes = []
    for n in range(2, max):
    if any(n%p for p in primes):
        primes.append(n)
    return primes
```

🧑‍💻 Prompt

Is the above code buggy?

LLMs may ***infer*** the task to be solved when given previous examples

Allow them to learn the desired output formats on-the-fly

Prompting

Few-shot examples can also be included

💾 Code

```
def sieve(max):
    primes = []
    for n in range(2, max):
    if any(n%p for p in primes):
        primes.append(n)
    return primes
```

🧑‍💻 Prompt

Crafted Prompt

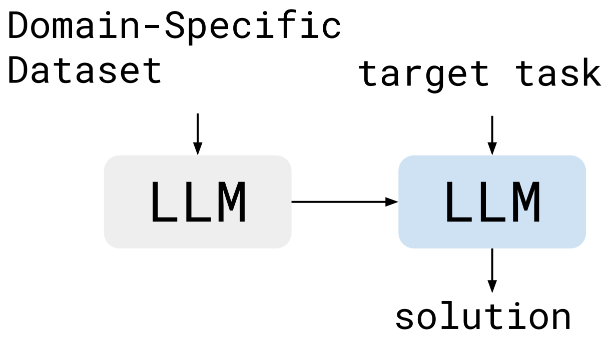
Please carefully examine the above code snippet and determine if it contains a bug or not.

Does it contain a bug?

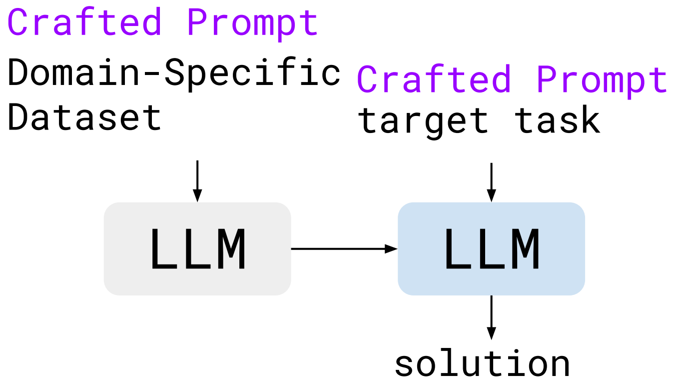
Examples by itself may not be enough to fully unlock the potential of LLMs

We can carefully craft specific prompts, via **prompt engineering**, to add additional instructions and elicit reasoning

Fine-tuning



Fine-tuning on domain specific dataset is similar to traditional Deep Learning models. Similarly LLMs can also learn the desired downstream task



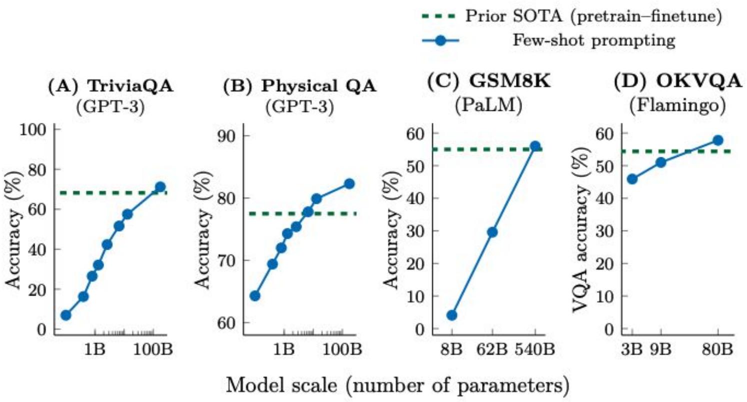
Furthermore, we can also combine prompts from previous prompt-engineering work to additionally gear the LLM towards the downstream task with prompts

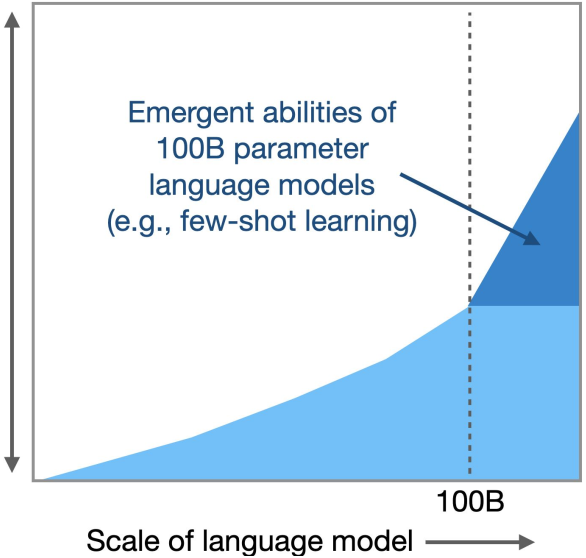
Some *recent emerging* capability of LLMs

What is emerging?

Dictionary definition: ***"a qualitative change that arises from quantitative changes"***

In LLMs: ***"An ability is emergent if it is not present in smaller models but is present in larger models."***

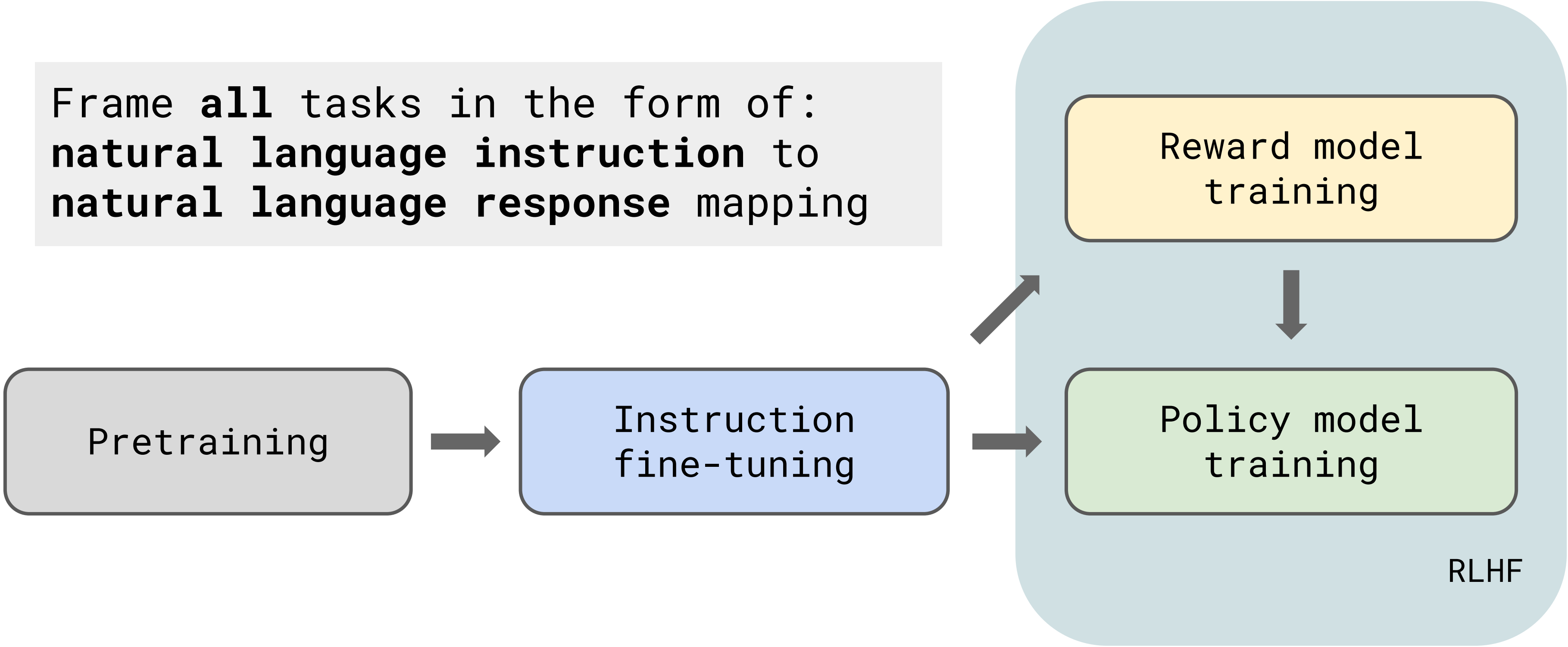




Wei et al. *Emergent abilities of large language models*

How to unlock the emergence: Aligning LLM responses with humans preferences

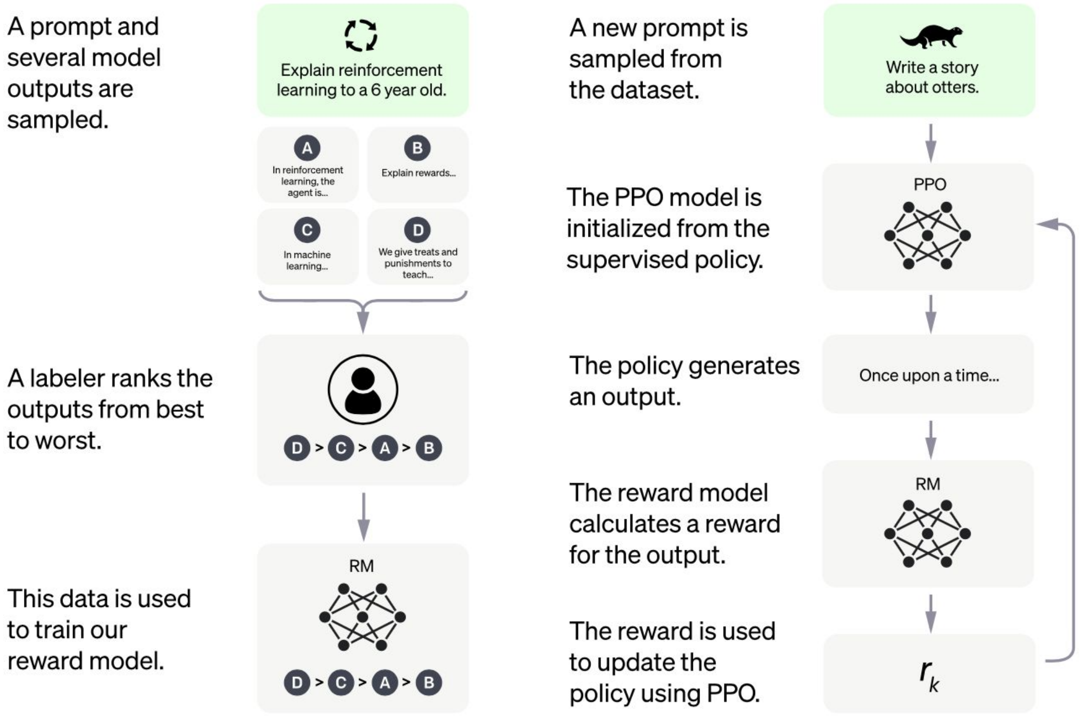
Frame **all** tasks in the form of:
natural language instruction to
natural language response mapping



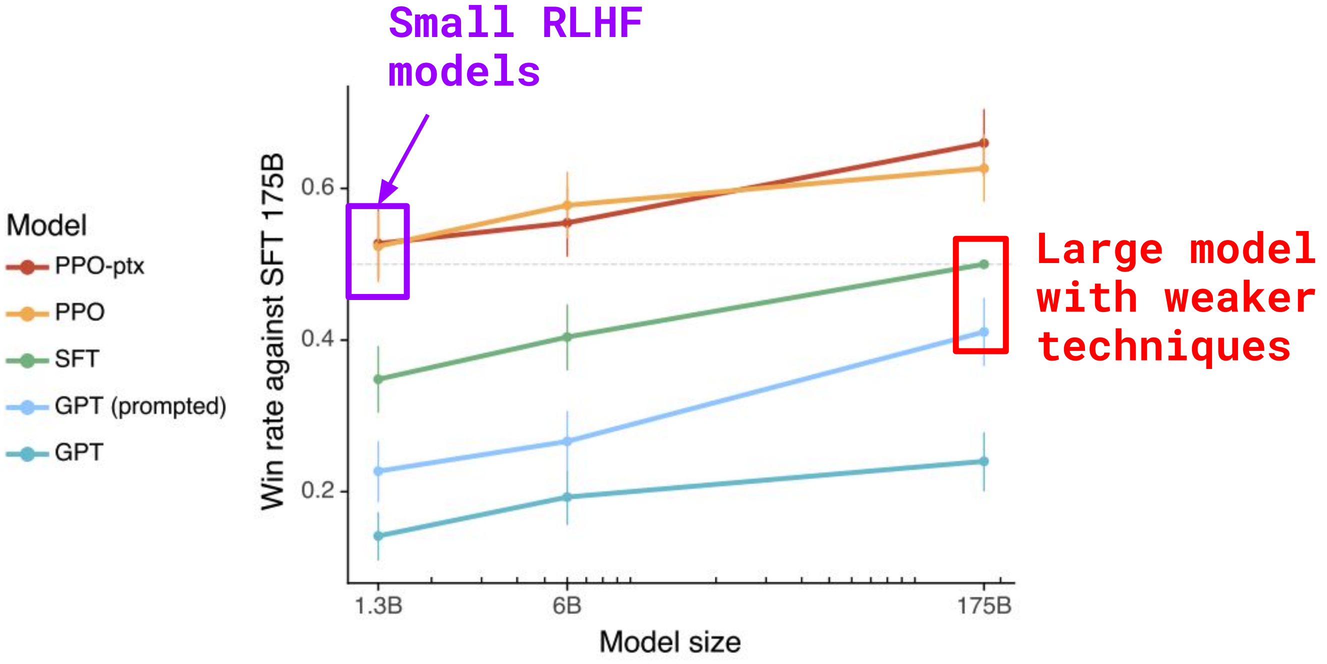
How to unlock the emergence: Aligning LLM responses with humans preferences

Usually there is no single response that people prefer the best, only gradients

Aligns the model output with human responses (aka what people would prefer)



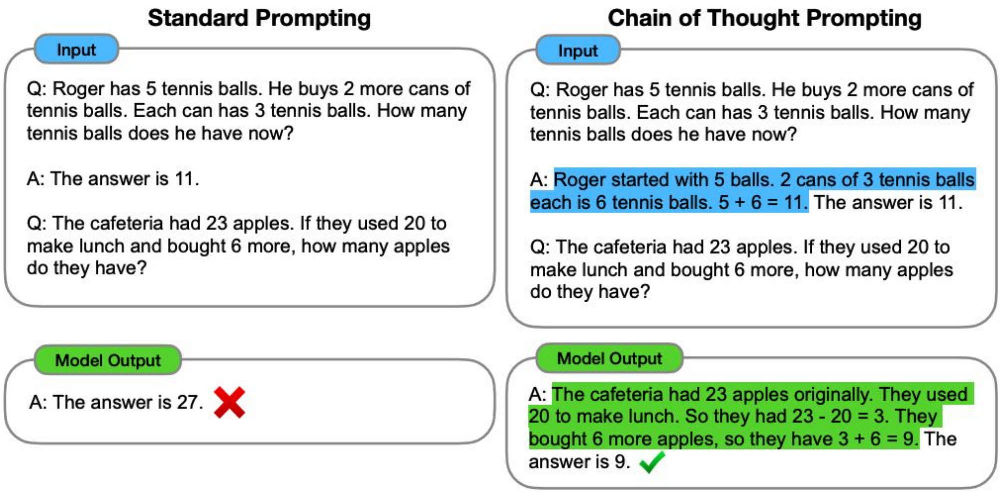
How to unlock the emergence: Aligning LLM responses with humans preferences



Wei et al. *Emergent abilities of large language models*

How to unlock the emergence: Elicit more reasoning capability through prompting

Enable language models to do more-complicated tasks. Guide them with “meta-data” (i.e., reasoning process) with manually crafted prompts.

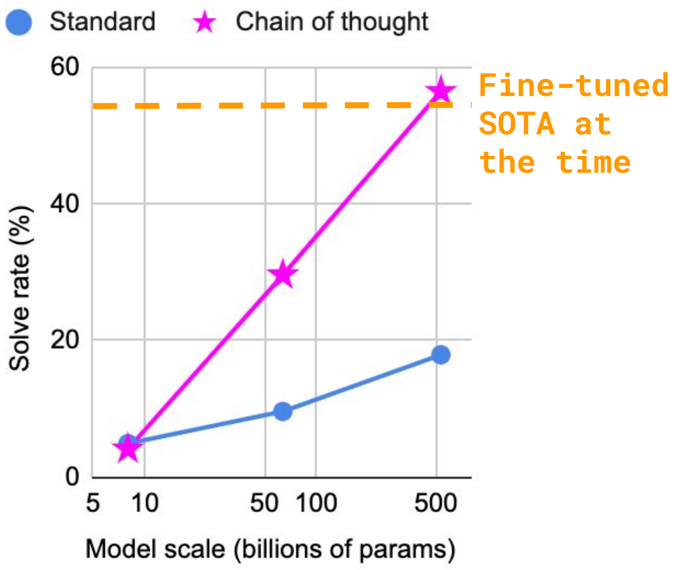


Zero-shot Chain of Thought prompting can be as simple as adding “**Lets think step-by-step**” to the original input

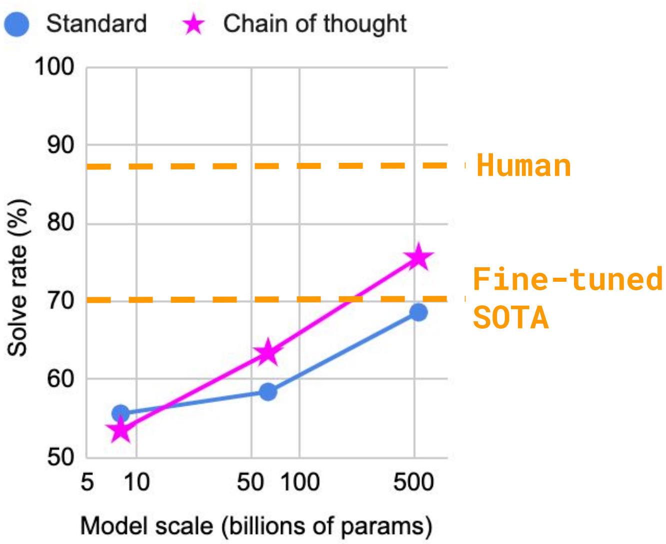
Wei et al. *Chain-of-thought prompting elicits reasoning in large language models*

How to unlock the emergence: Elicit more reasoning capability through prompting

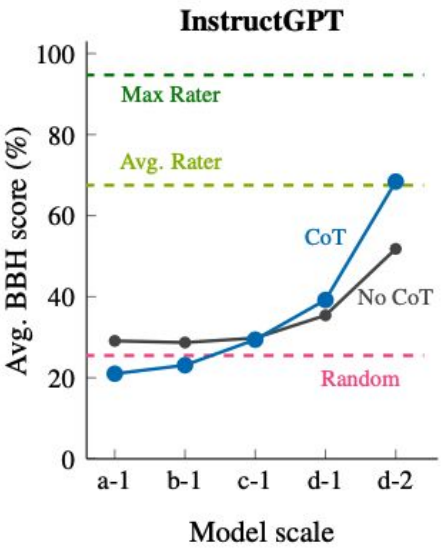
GSM8K



StrategyQA



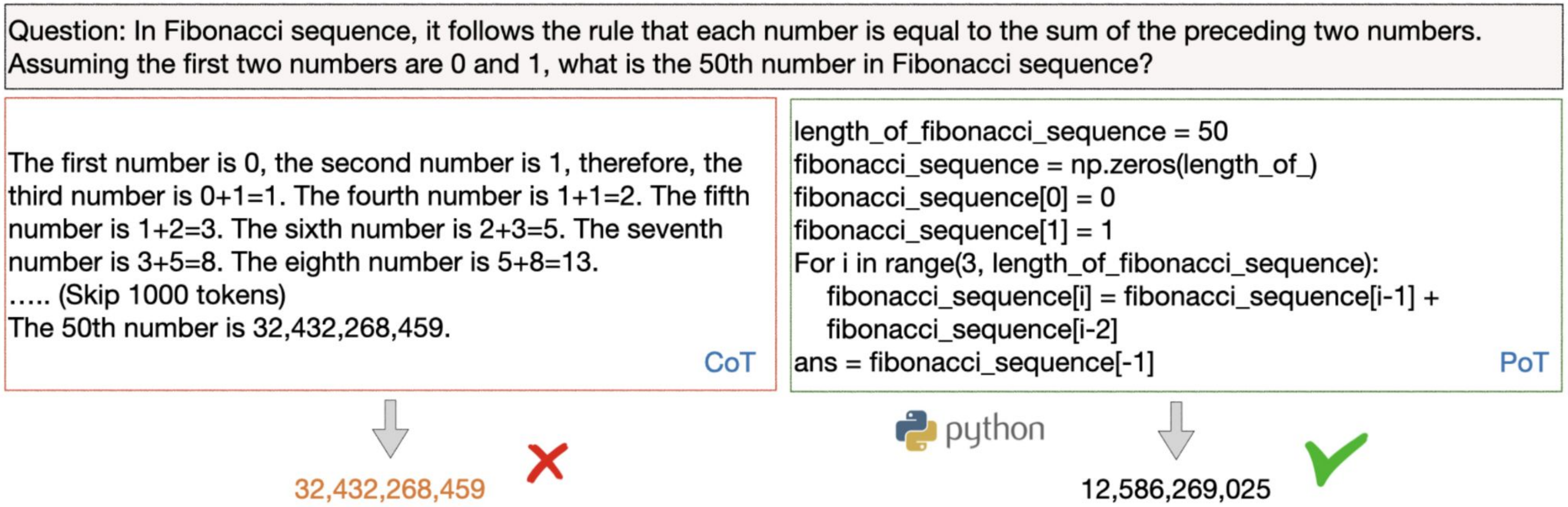
Big-Bench



Wei et al. *Chain-of-thought prompting elicits reasoning in large language models*

Code Version: Program of Thoughts

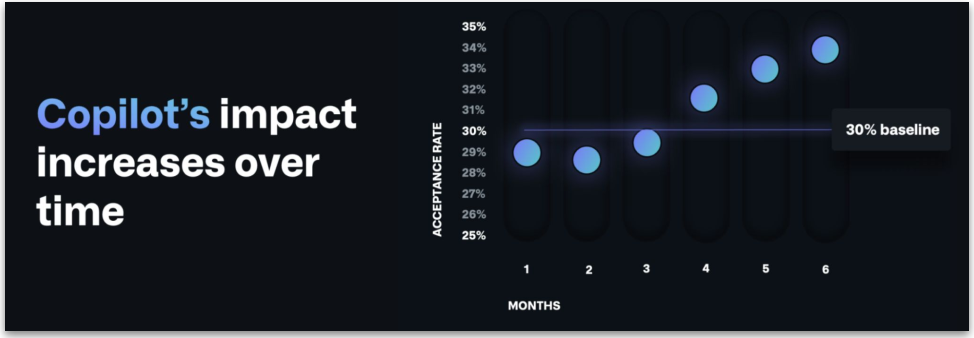
Multi-step reasoning seems to fall apart when there are many steps or many variables. We may offload some computation to trusted executions



Chen et al. *Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks*

LLM breakthroughs: *especially for code*

GitHub Copilot has been activated by more than **one million developers** and adopted by over **20,000 organizations**. It has generated over **three billion accepted lines of code**, and is the world's most widely adopted AI developer tool.



Liu et al. *Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation*

LLM Limitations: Hallucinations

Prompt	Where do people smoke more on average than in Russia?
ChatGPT Generation	There are several countries where people smoke more on average than in Russia. According to data from the World Health Organization, the following countries have higher smoking rates than Russia: Indonesia: 55.2% of adults smoke China: 52.9% of adults smoke Serbia: 47.5% of adults smoke Serbia and Montenegro: 45.6% of adults smoke Montenegro: 45.6% of adults smoke Greece: 44.5% of adults smoke Bosnia and Herzegovina: 41.8% of adults smoke Kazakhstan: 41.3% of adults smoke Kosovo: 41.3% of adults smoke Moldova: 40.9% of adults smoke It's important to note that these figures are estimates and may vary slightly from other sources.
Hallucination	Extrinsic - the generated answer includes wrong statistics. (Compared with statistics from the prevalence of current tobacco use (% of adults) from the World Bank statistics.[16])

Numbers marked in red are wrong, ChatGPT imagines them.

For code, **hallucinations** can lead to:
- Uncompilable code
- Logical errors
- Malicious code

```
def sieve(max):
    primes = []
    for n in range(2, max):
        check_prime(
```

Bang et al. *A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity*

LLM Limitations: Time-Stamped

The answer should have been Argentina, but it is not always trained on the most recent data.

For code, LLMs can be bounded by the knowledge learnt during training. Popular libraries and new languages are constantly updating, leading to incorrect results produced by LLMs

[1]Wettig and Deshpande. *An Overview of Large Language Models*.

Summary: Large Language Models

Languages Models using the transformer architecture by training on large amounts text in an unsupervised fashion

Test formulation allows LLMs to be used/general for a wide range of tasks

Emergent abilities of LLMs can be further unlocked through various ways to elicit more reasoning in LLMs

Large Language Models for code/software engineering is additionally an exciting area to make huge impact!