Transformers
the why the what and the how

What to look forward to

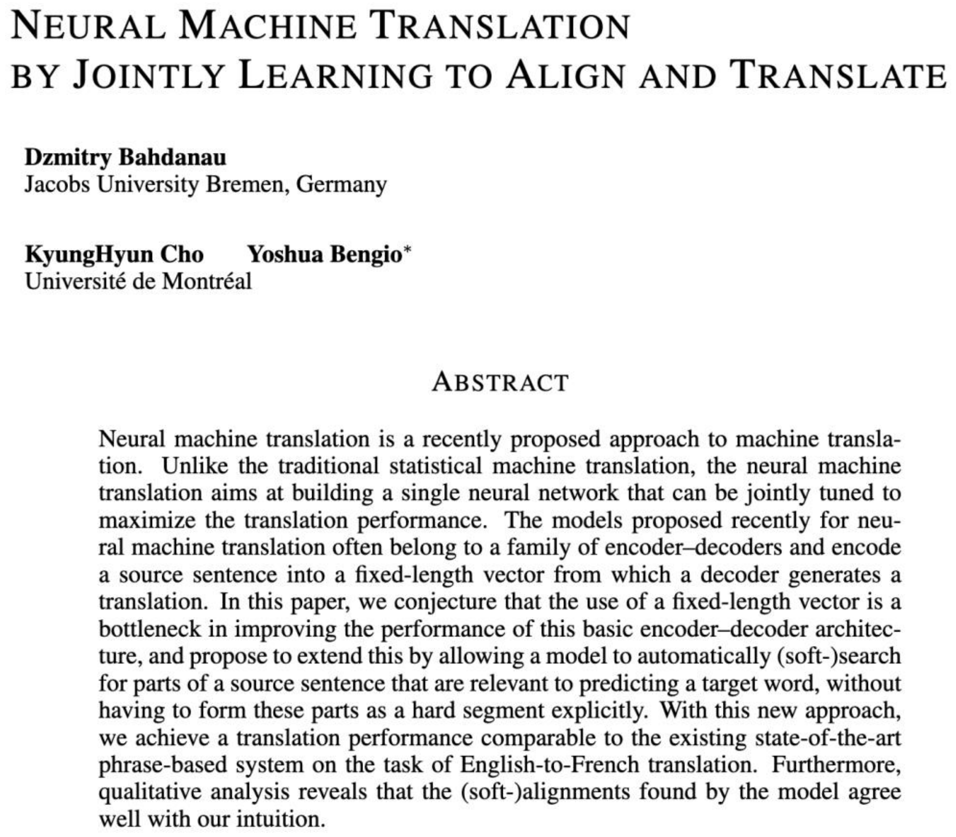
- (15min) Recap
- (30 min) Decoder only models
- (10 min)Break
- (30 min) Applications
  - molecules, language, reinforcement learning
- (25 min) How do we train one?
- (10 min)Break
- (15 min) Different stages of training
  - Pre-training, mid training, post training
- (15 min) LLM agents
  - How do ChatGPT, Claude and Gemini actually work?
- (15 min) Advanced Research topics
  - Mechanistic interpretability, post-training, pre-training

Recap of transformers

- The main driver of transformers is self-attention
    - More on this later
- Why are they better than RNNs?
    - Better at modeling long sequences as well as more efficient training
- Generative models
    - Decoder based models

Where does attention even come from?

Attention the core algorithm that drives LLMs
Was actually invented by Dima (Dzmitry) Bahdanau in Montreal



NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

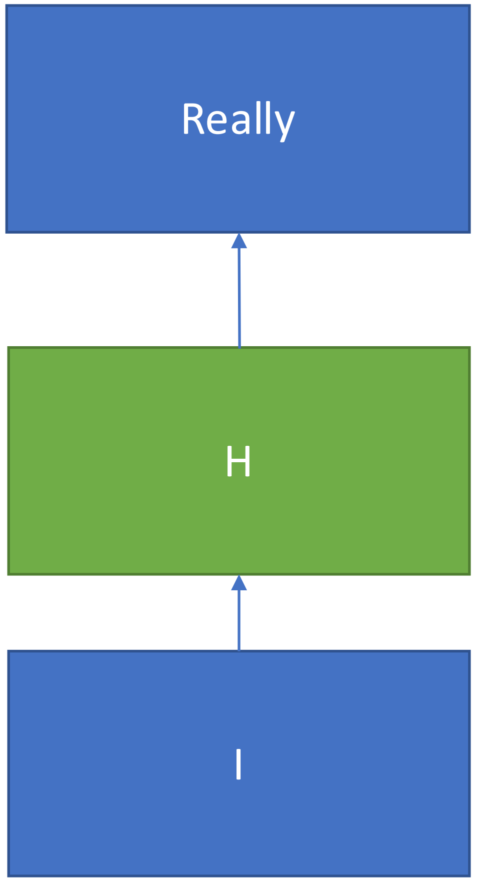
Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho Yoshua Bengio*
Université de Montréal

ABSTRACT

Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance. The models proposed recently for neural machine translation often belong to a family of encoder–decoders and encode a source sentence into a fixed-length vector from which a decoder generates a translation. In this paper, we conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and propose to extend this by allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. With this new approach, we achieve a translation performance comparable to the existing state-of-the-art phrase-based system on the task of English-to-French translation. Furthermore, qualitative analysis reveals that the (soft-)alignments found by the model agree well with our intuition.

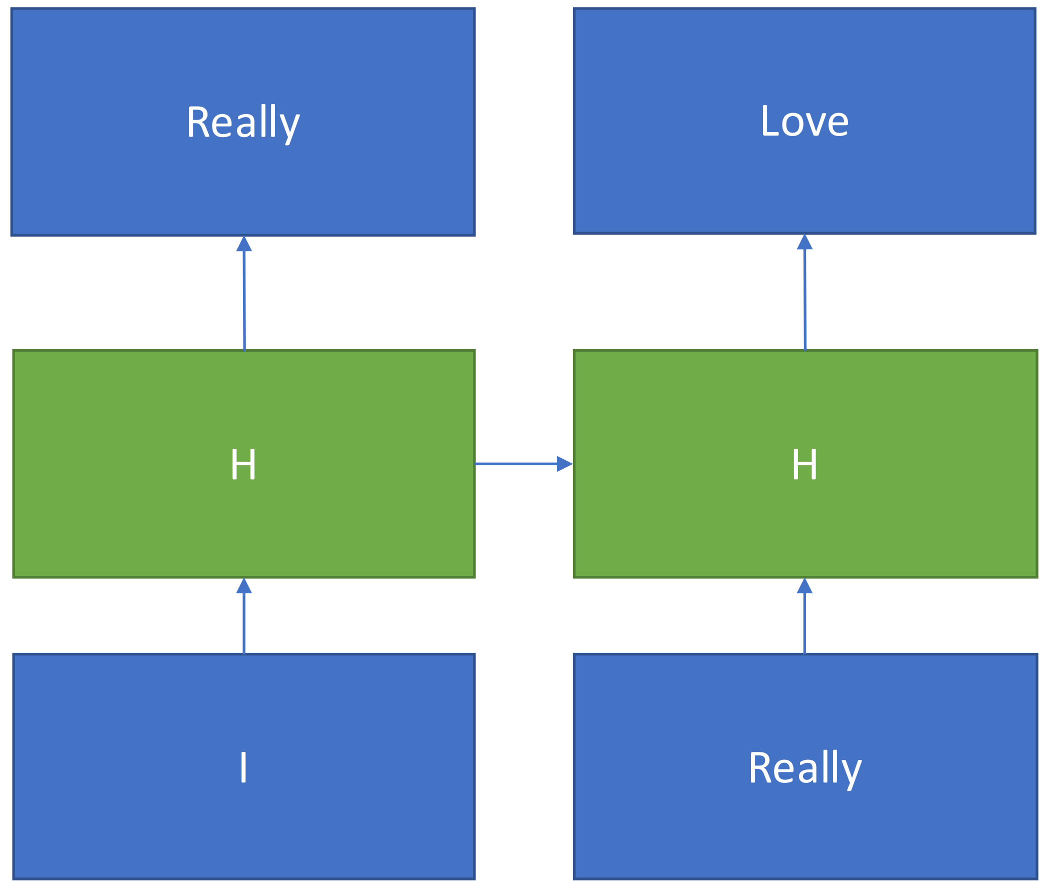
Why is attention better than recurrence?



RNNs are non parallelizable

If we want to train, we must compute each token one at a time

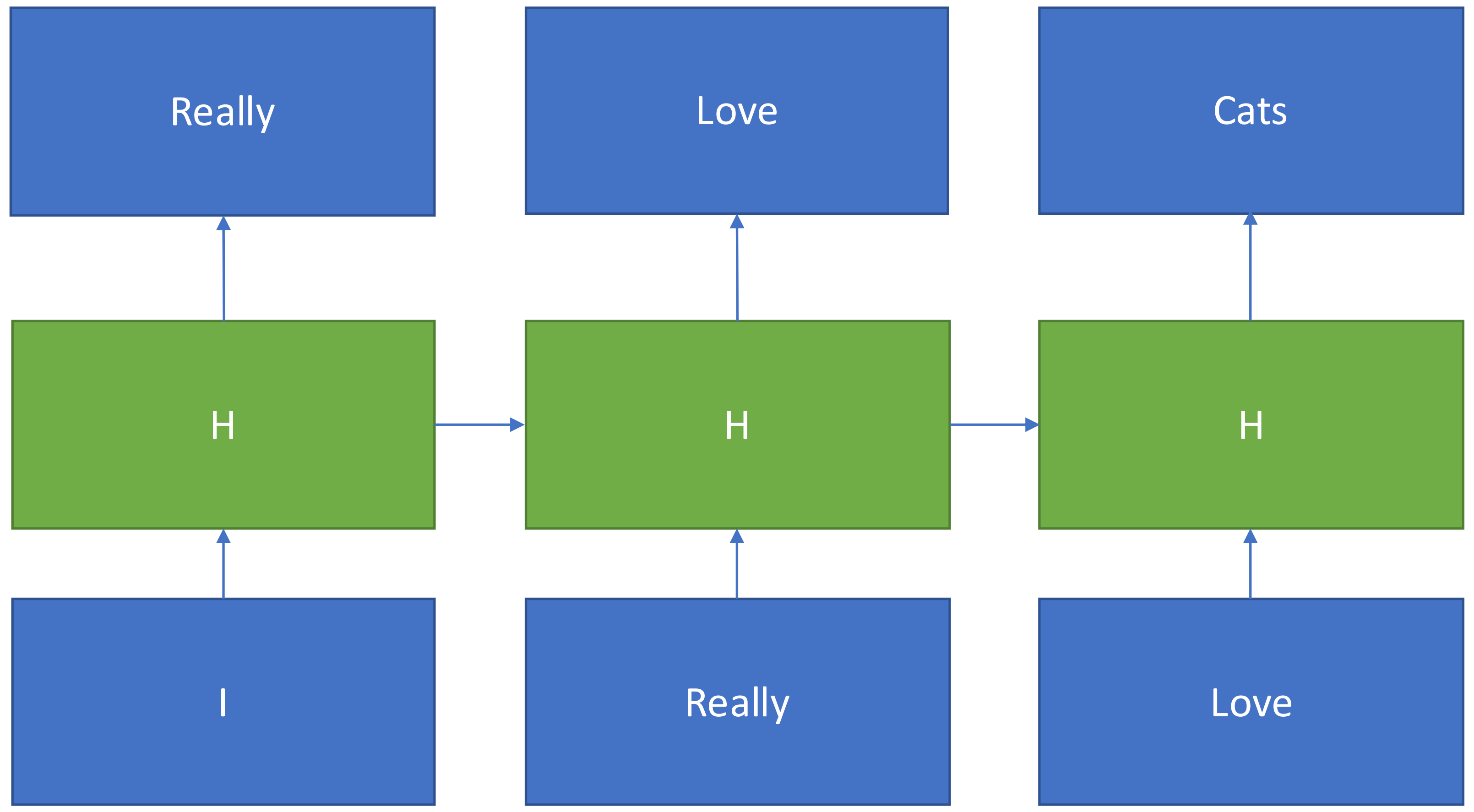
Why is attention better than recurrence?



RNNs are non parallelizable

If we want to train, we must compute each token one at a time

Why is attention better than recurrence?

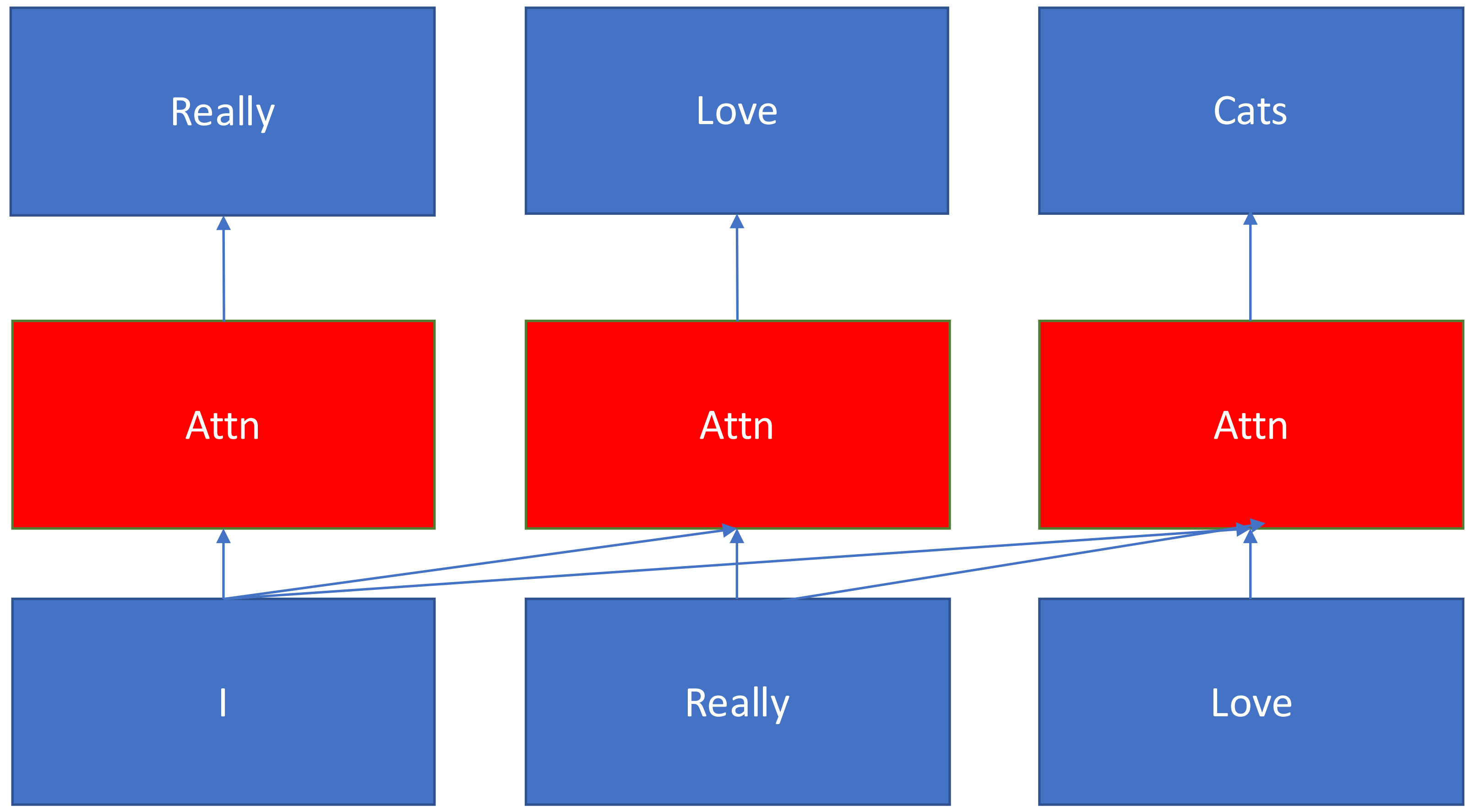


RNNs are non parallelizable

If we want to train, we must compute each token one at a time

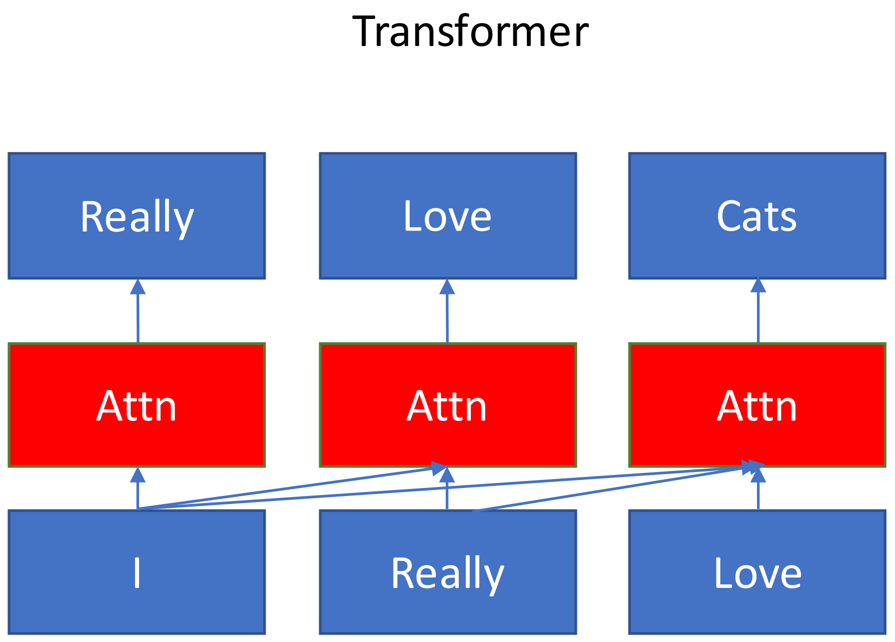
This is slow, makes consuming large amount of data much harder

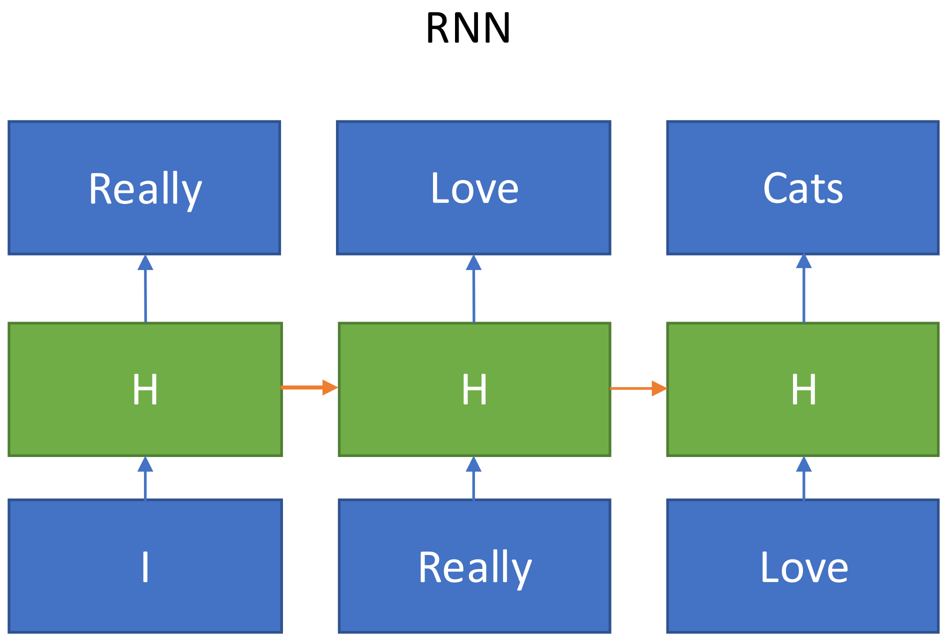
Why is attention better than recurrence?



Transformers can process a whole sequence at once. This makes training extremely parallelizable

Why is attention better than recurrence?





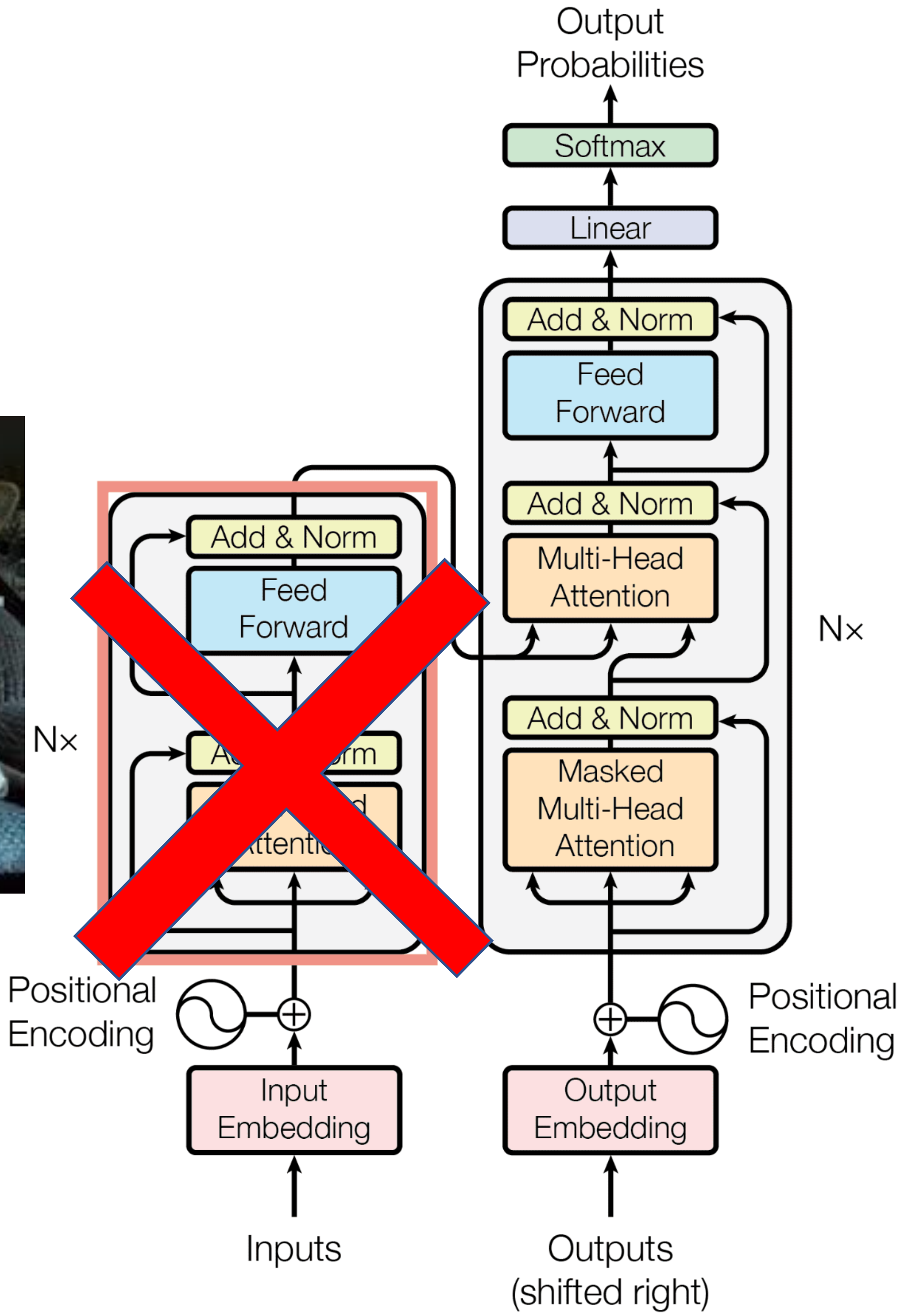
These (now) orange arrows are the problem. As having to wait for the output of prior inputs to continue training does not allow us to break up the computation

Ok so attention is amazing are there any drawbacks?

- Yes attention is slower at prediction (when producing tokens) compared to RNNs
- This is because for transformers it must attend to the entire sequence before outputting the next token
- In RNNs we can just look at the current token since all prior information is encapsulated in the hidden state (H in prior slides)
- This makes inference on transformers be O(n^2) vs O(n) in RNNs

Decoder only models

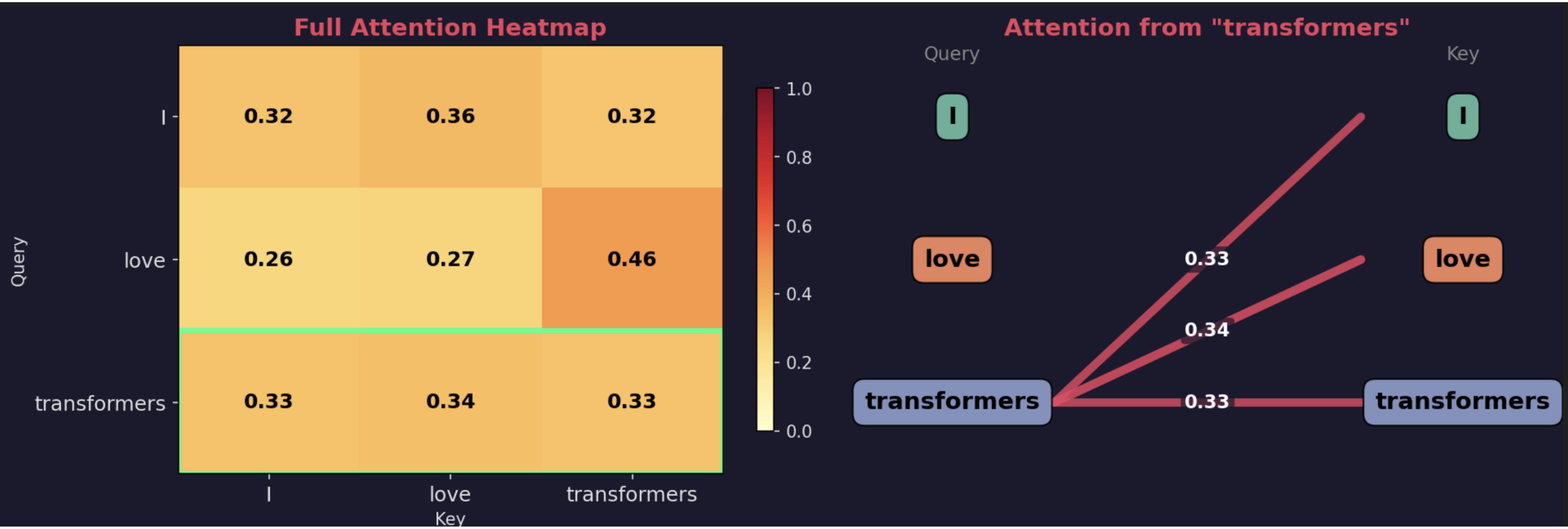
- These days (almost) all text based transformers are decoder only.
- The simple reason is the transformer decoder can already read prior words so there is no need to encode them first
- This simplifies the model which almost always a good thing



Recap – Self attention

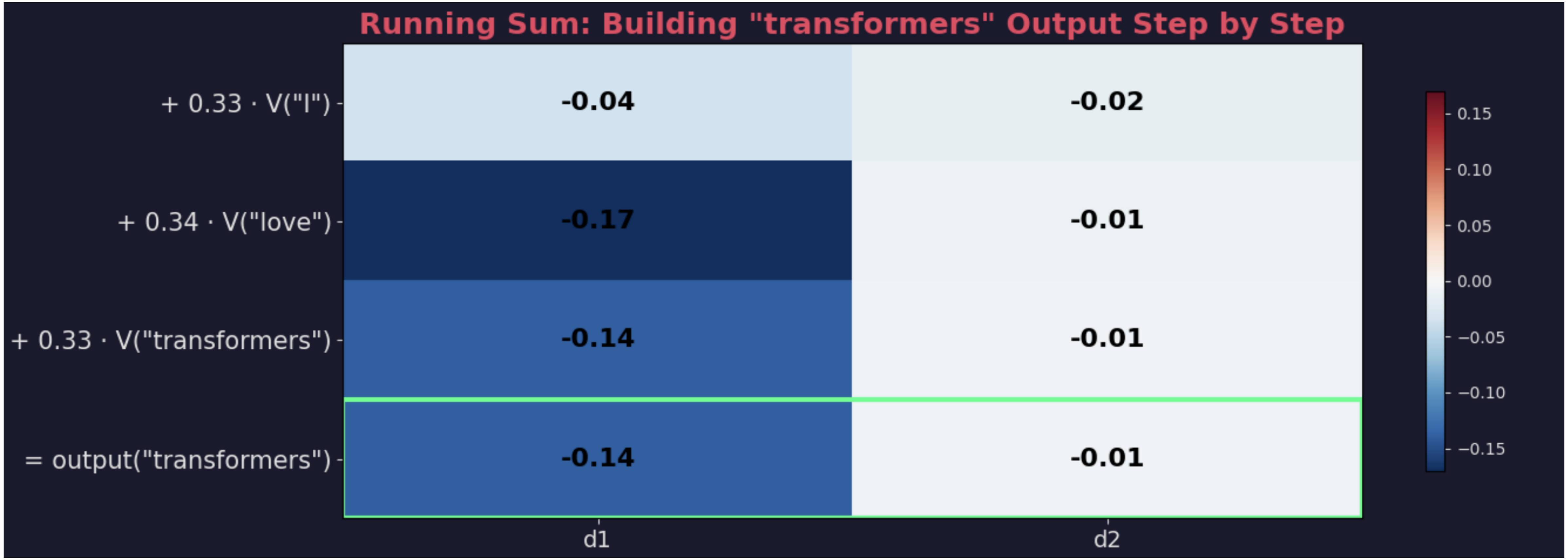
Lets visualize the math step by step

$$\text{Attention}(Q,K,V) = \boxed{\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)}V$$



Recap – Self attention

$$\text{Attention}(Q, K, V) = \boxed{\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V}$$

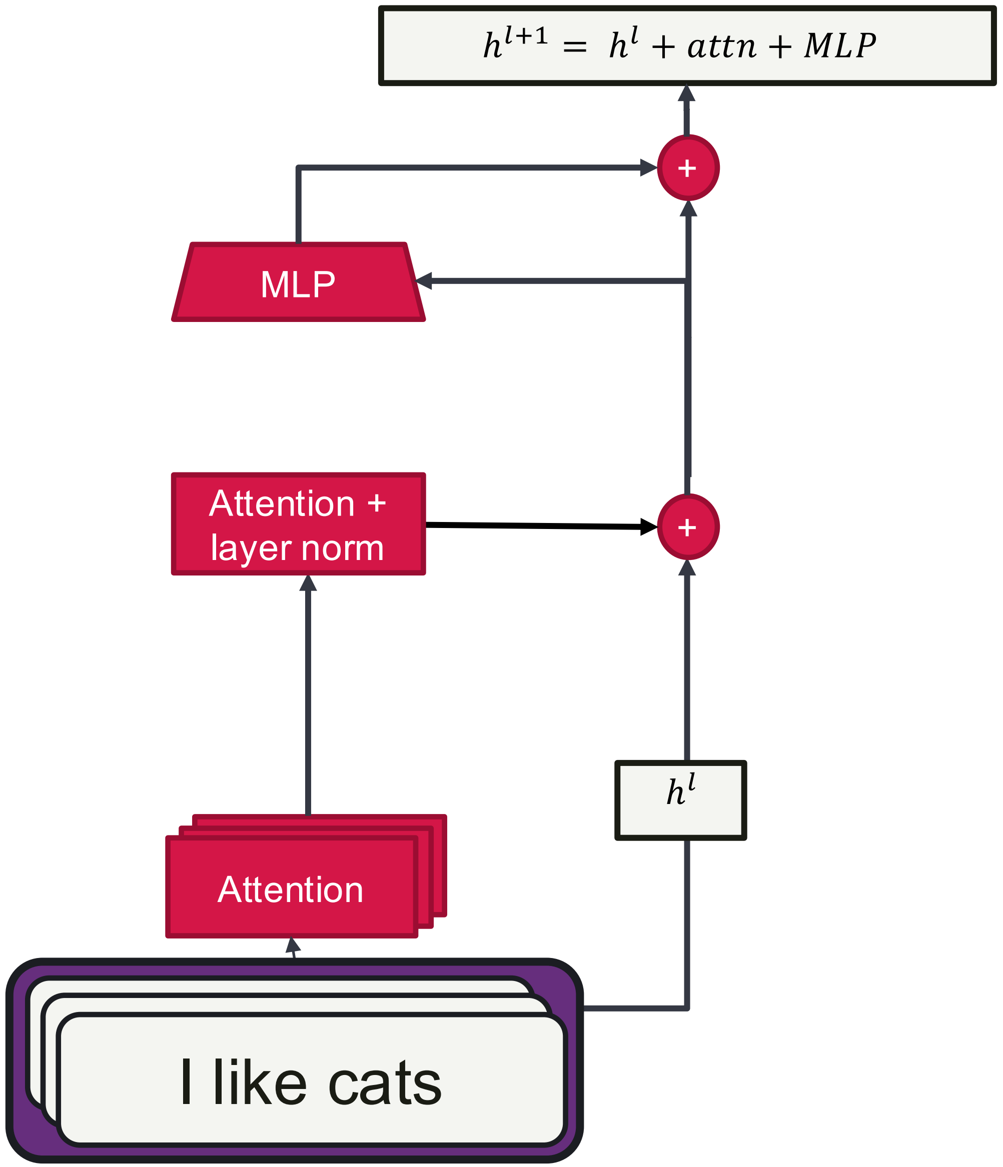

Running Sum: Building "transformers" Output Step by Step

	d1	d2
+ 0.33 · V("I")	-0.04	-0.02
+ 0.34 · V("love")	-0.17	-0.01
+ 0.33 · V("transformers")	-0.14	-0.01
= output("transformers")	-0.14	-0.01

So in simple terms what is attention doing?

- Given a query q look at the keys K
- Check how important each key is for the query
- Update the representation of the query as the mixture of these representations

How does a transformer use attention



Transformers are just attention based residual networks

Residual networks help learn as they largely solve the vanishing gradient problem

We use layer norm for training stability and then pass back to the MLP to update the representations

The transformer is ”simple”

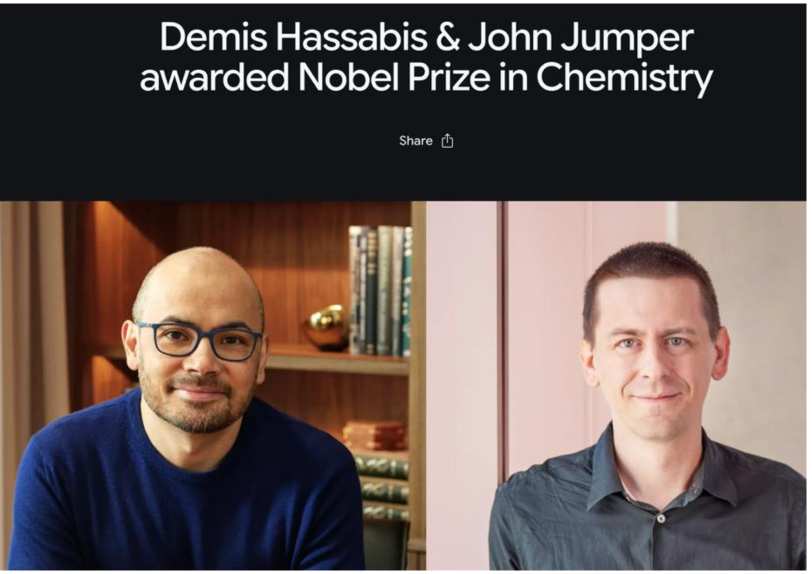
- Using this simple mechanism the transformer has become the dominant architecture in almost all deep learning
- The main pros are its training **parallelizability**
- Ability to model **long sequences**
- Resnet structure aids with vanishing gradient problem providing **Stability**

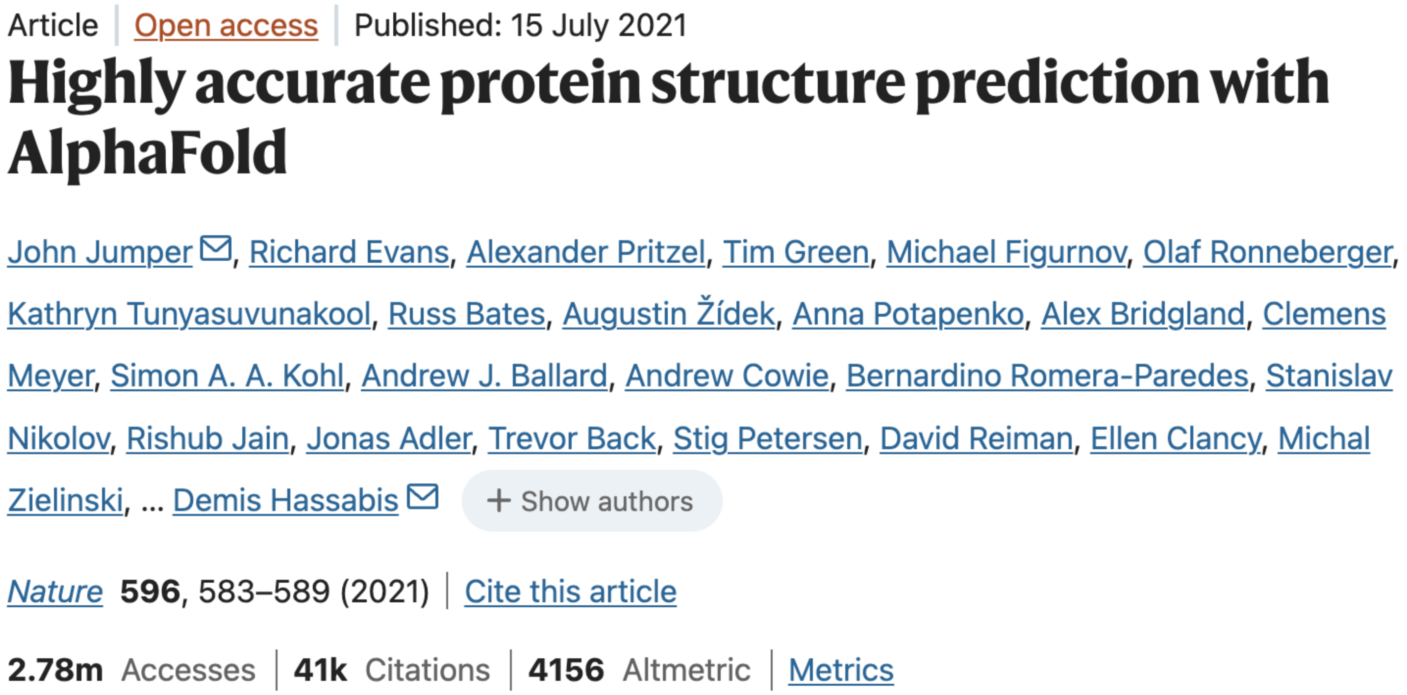
AI for science

One of the biggest problems in biology is the protein folding problem

This was largely "solved" by AlphaFold in 2021

Timeline of a breakthrough
In November 2020, AlphaFold was recognised as a solution to the 50-year "protein-folding problem".


Demis Hassabis & John Jumper awarded Nobel Prize in Chemistry
Share


Article | Open access | Published: 15 July 2021

Highly accurate protein structure prediction with AlphaFold

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, ... Demis Hassabis + Show authors

Nature **596**, 583–589 (2021) | Cite this article

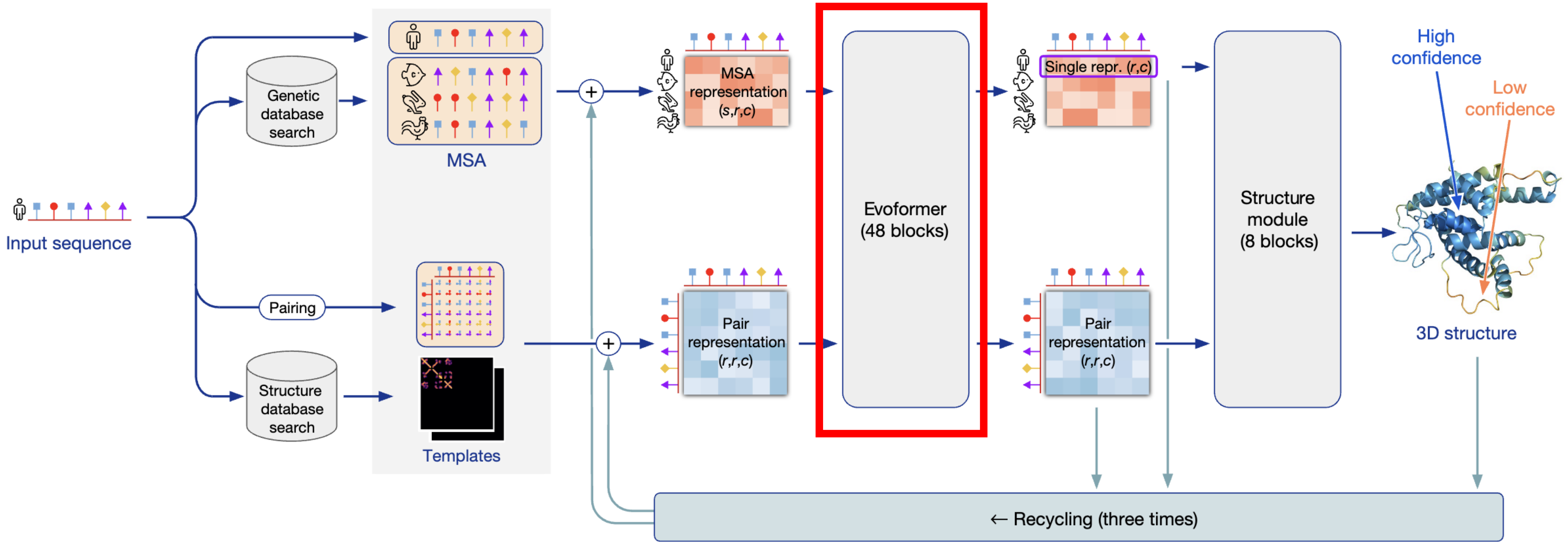
2.78m Accesses | **41k** Citations | **4156** Altmetric | Metrics

Highly accurate protein structure prediction with AlphaFold, Google team 2021 (https://www.nature.com/articles/s41586-021-03819-2)

AI for science

One of the biggest problems in biology is the protein folding problem

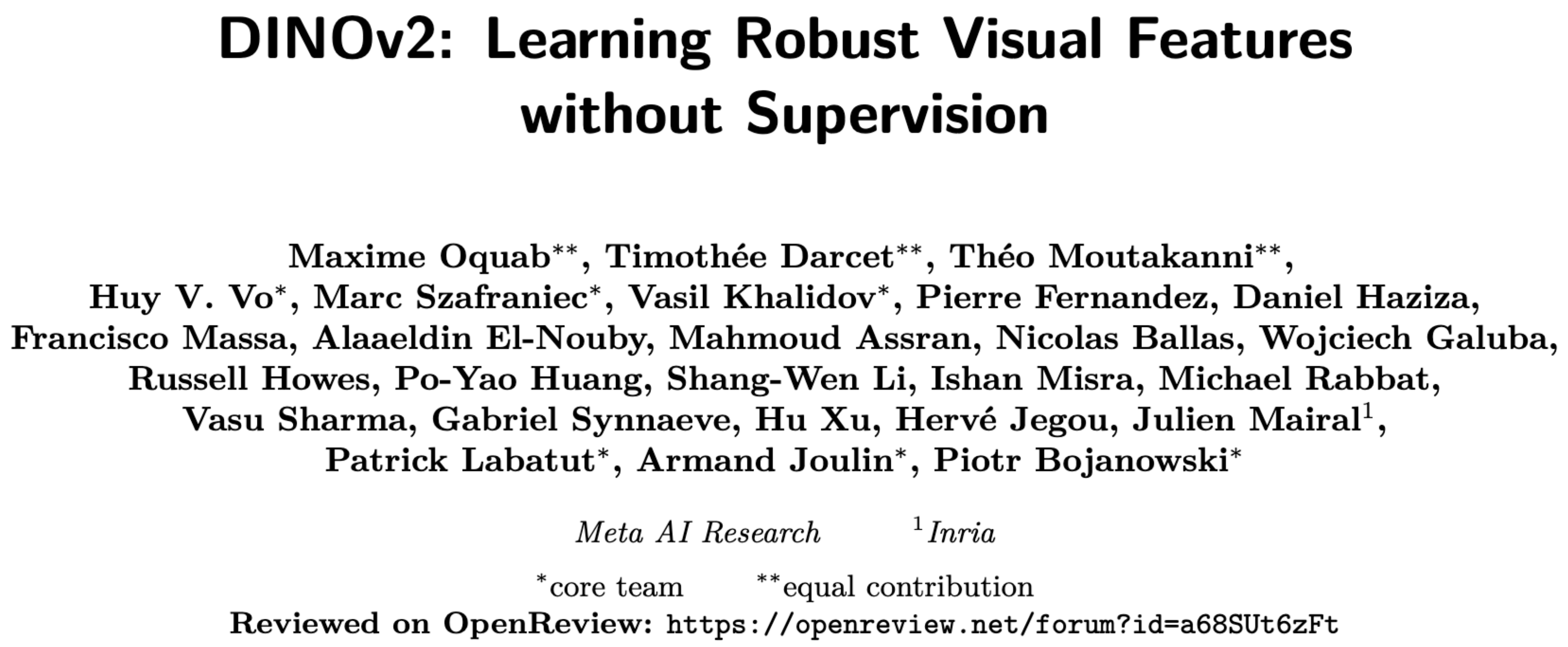
This was largely "solved" by AlphaFold in 2021



Input sequence
Genetic database search
Pairing
Structure database search
MSA
Templates
MSA representation (s,r,c)
Pair representation (r,r,c)
Evoformer (48 blocks)
Single repr. (r,c)
Pair representation (r,r,c)
Structure module (8 blocks)
High confidence
Low confidence
3D structure
← Recycling (three times)

Highly accurate protein structure prediction with AlphaFold, Google team 2021 (https://www.nature.com/articles/s41586-021-03819-2)

Computer Vision



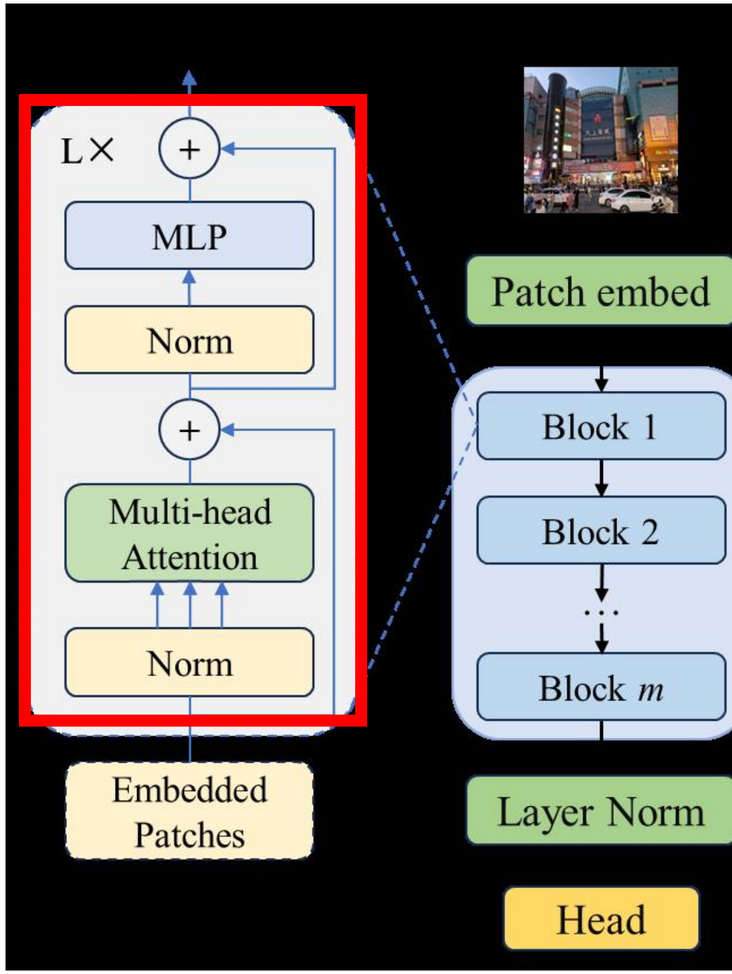
DINOv2: Learning Robust Visual Features without Supervision

Maxime Oquab**, Timothée Darcet**, Théo Moutakanni**,
Huy V. Vo*, Marc Szafraniec*, Vasil Khalidov*, Pierre Fernandez, Daniel Haziza,
Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba,
Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat,
Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal[1],
Patrick Labatut*, Armand Joulin*, Piotr Bojanowski*

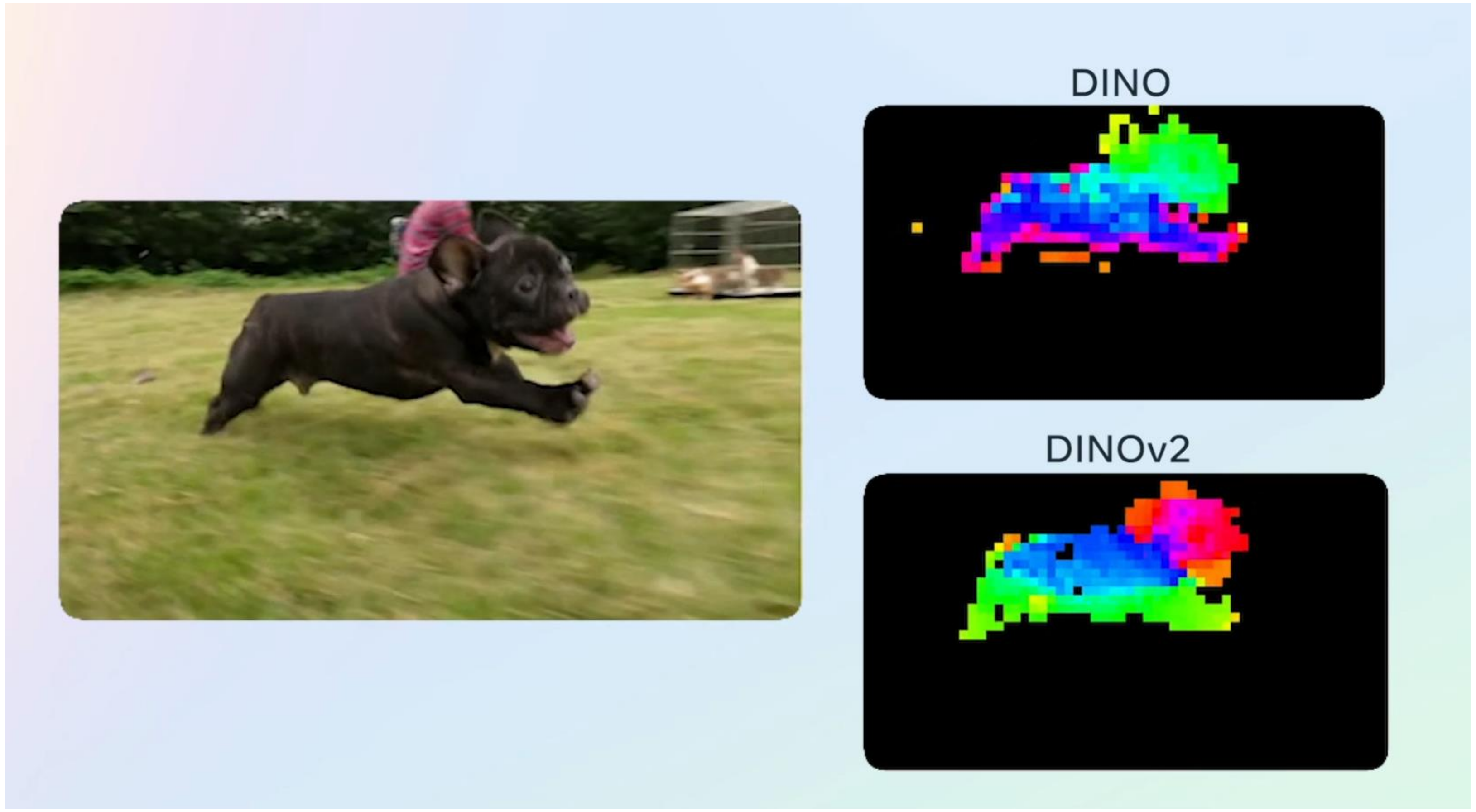
Meta AI Research [1] *Inria*

*core team **equal contribution

Reviewed on OpenReview: `https://openreview.net/forum?id=a68SUt6zFt`



Computer Vision



There are a lot more!

- Lets take 10 minutes to discuss
- Look up applications you find the most interesting
- Submit them to the link below and we’ll briefly discuss them!

How do we train a transformer?

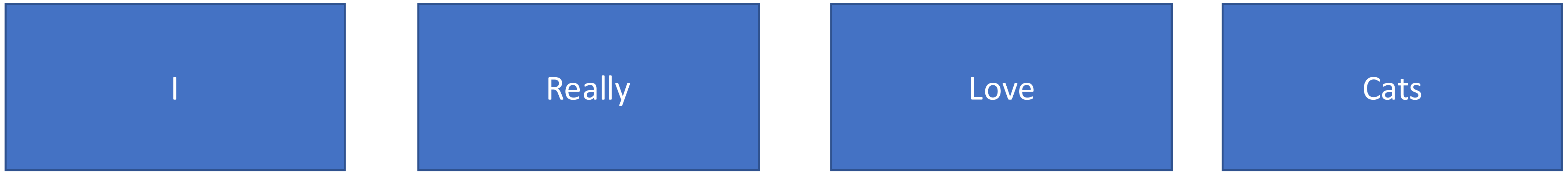
- For the remaining of the lecture we will stick to **decoder** only transformers
- Discuss **teacher forcing**
- The negative log likelihood loss
- Brief introduction to perplexity
- Brief comment on “Scaling laws”

Teacher forcing

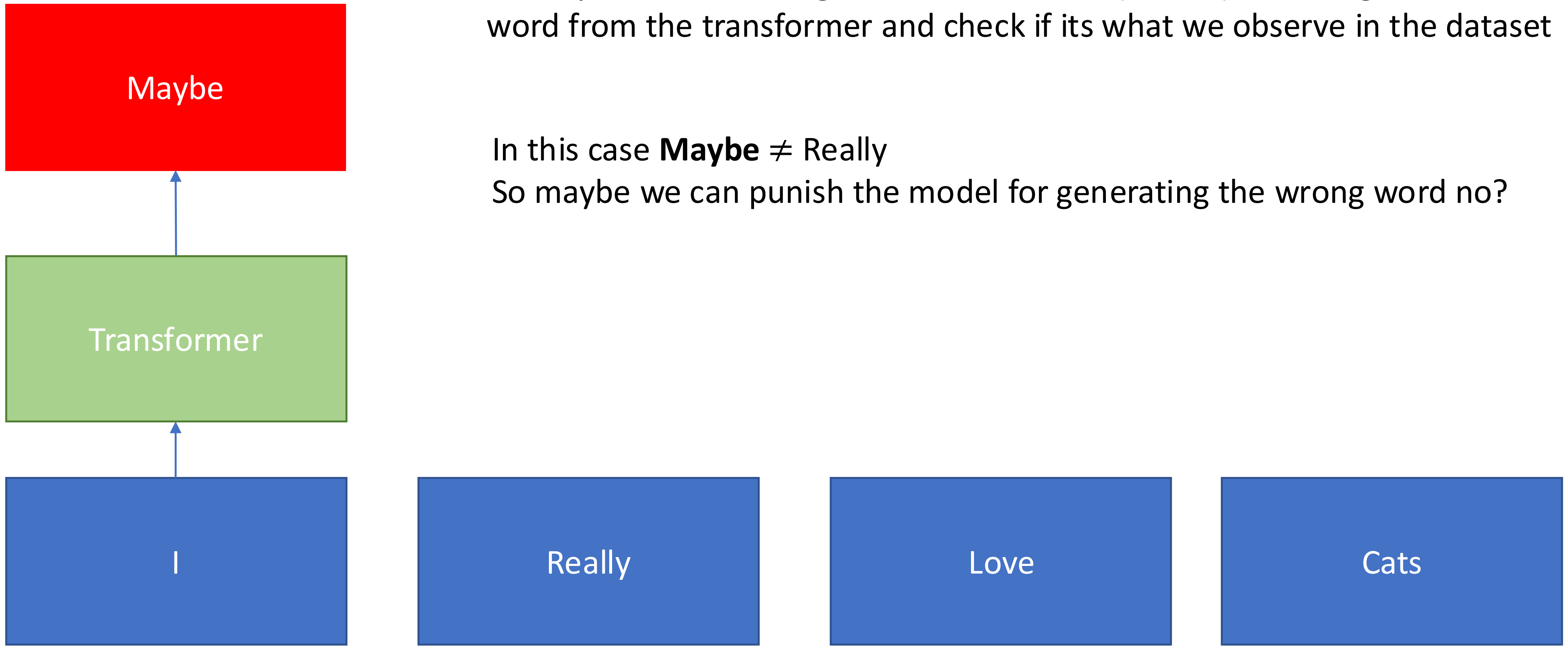
- Lets take this step by step
- If we wanted to train a transformer and we have the entire corpus of the web available to us how can we do this?

Teacher forcing

Lets say we have 3 words and want to teach a transformer this sequence lets first go over the “naïve” way to do this



Teacher forcing

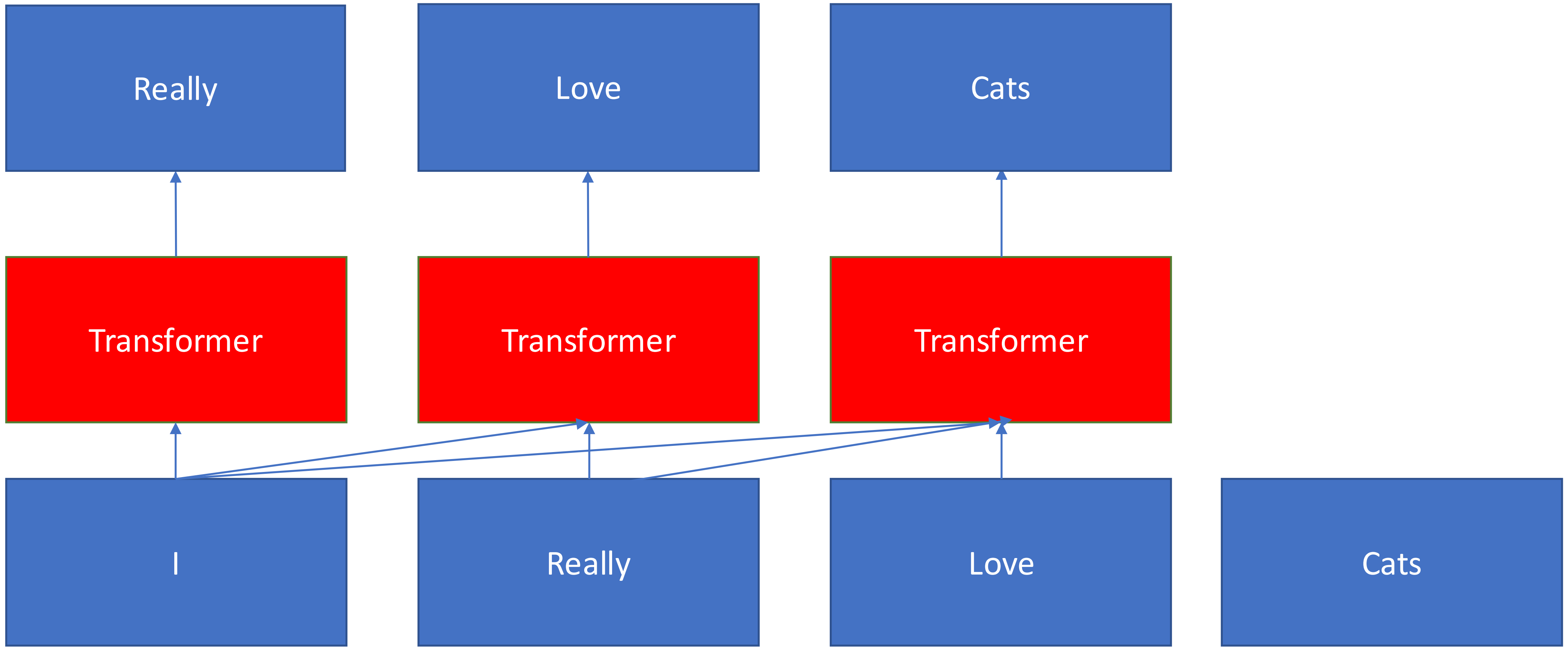


Naively since I want to generate new words (tokens), we can generate a new word from the transformer and check if its what we observe in the dataset

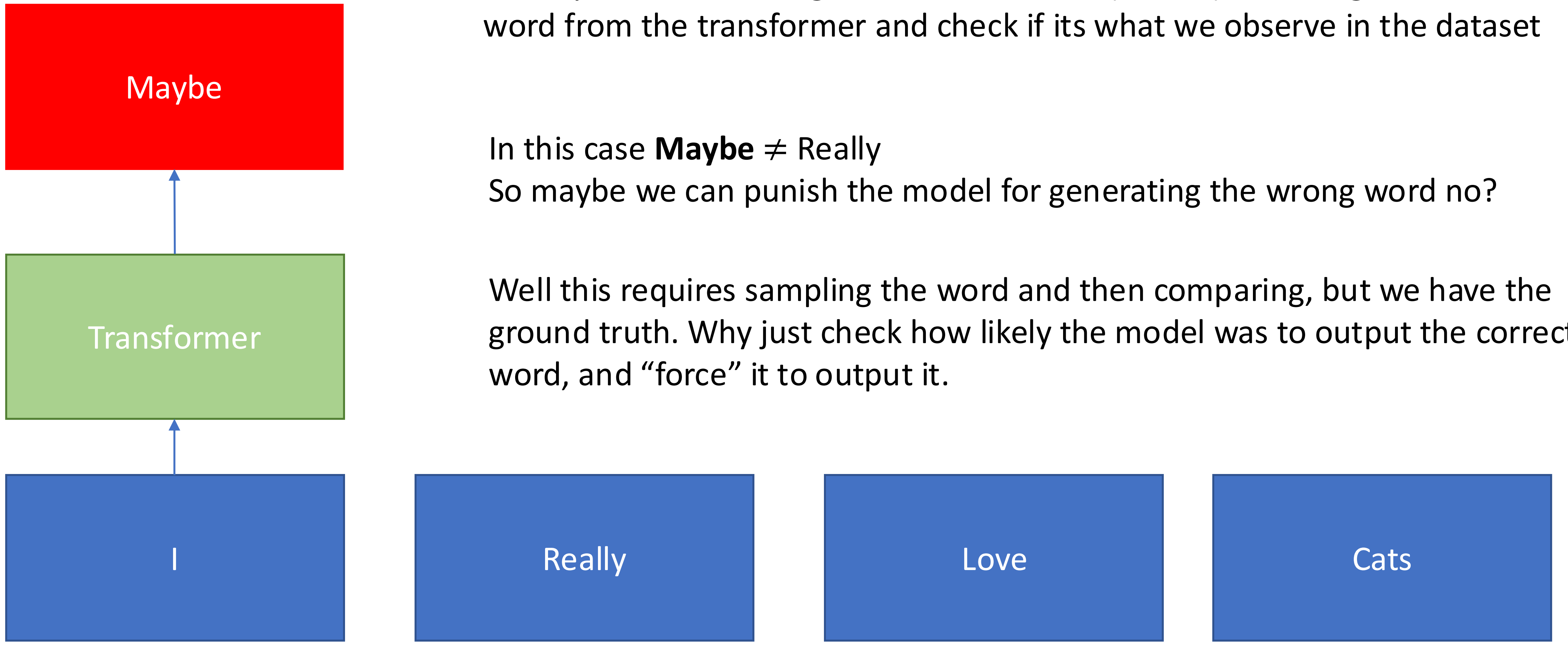
In this case **Maybe** ≠ Really
So maybe we can punish the model for generating the wrong word no?

Teacher forcing

Lets say we have 3 words and want to teach a transformer this sequence



Teacher forcing

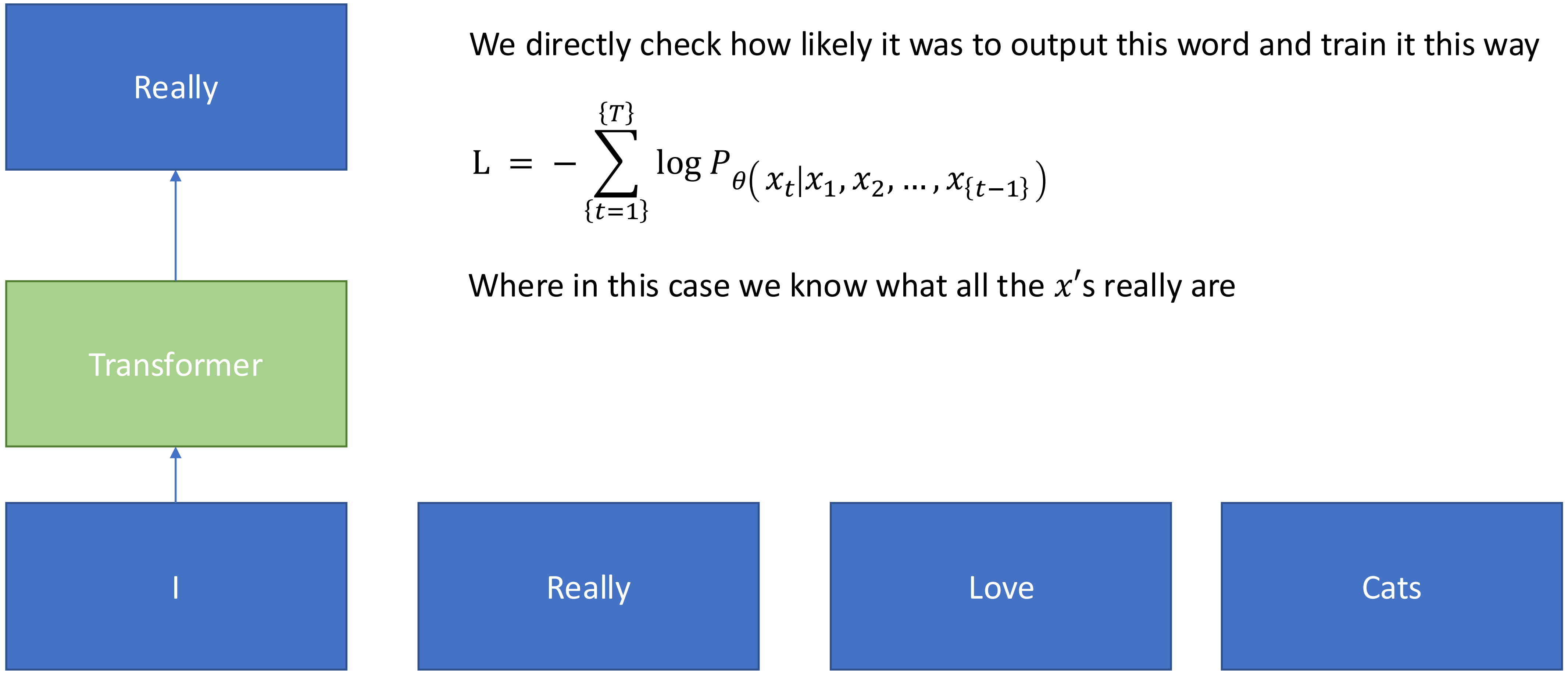


Naively since I want to generate new words (tokens), we can generate a new word from the transformer and check if its what we observe in the dataset

In this case **Maybe** $\neq$ Really
So maybe we can punish the model for generating the wrong word no?

Well this requires sampling the word and then comparing, but we have the ground truth. Why just check how likely the model was to output the correct word, and “force” it to output it.

Teacher forcing



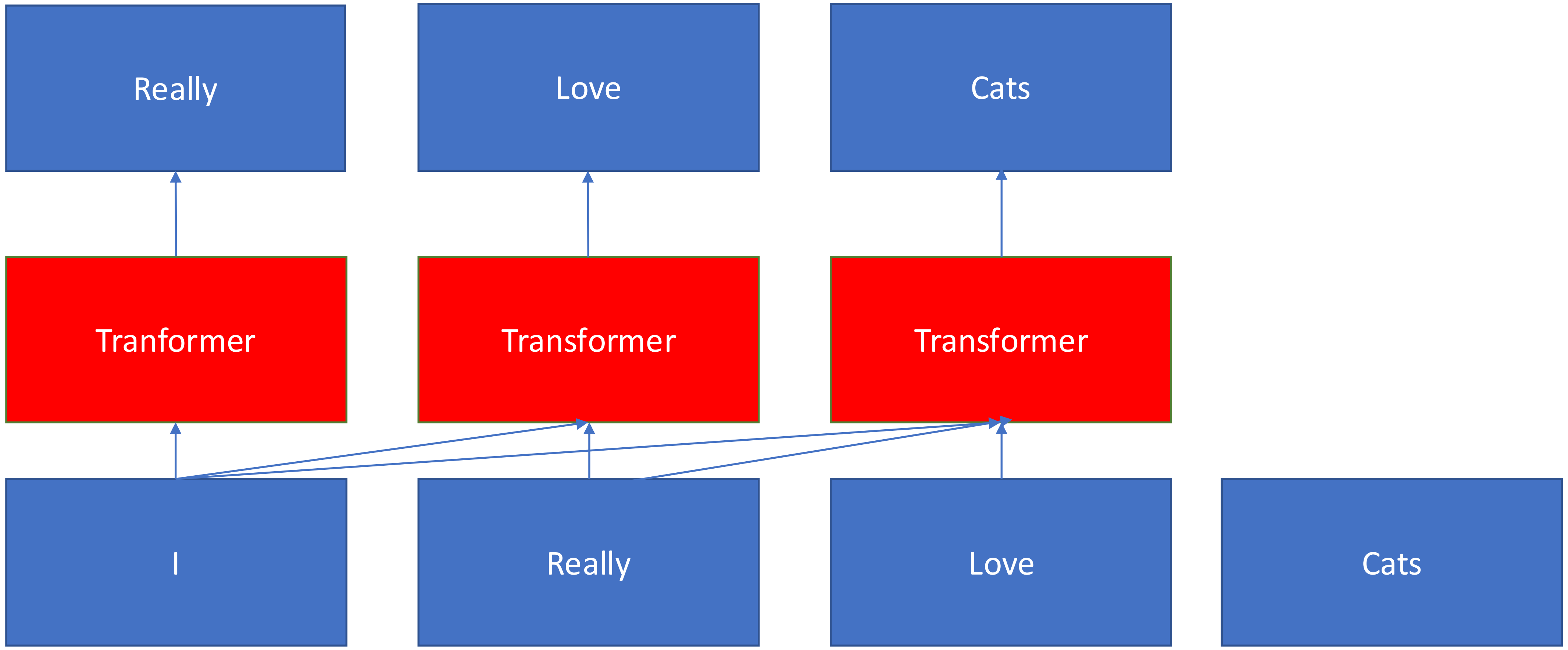
We directly check how likely it was to output this word and train it this way

$$\mathrm{L} = -\sum_{\{t=1\}}^{\{T\}} \log P_{\theta}\left(x_t | x_1, x_2, \ldots, x_{\{t-1\}} \right)$$

Where in this case we know what all the x's really are

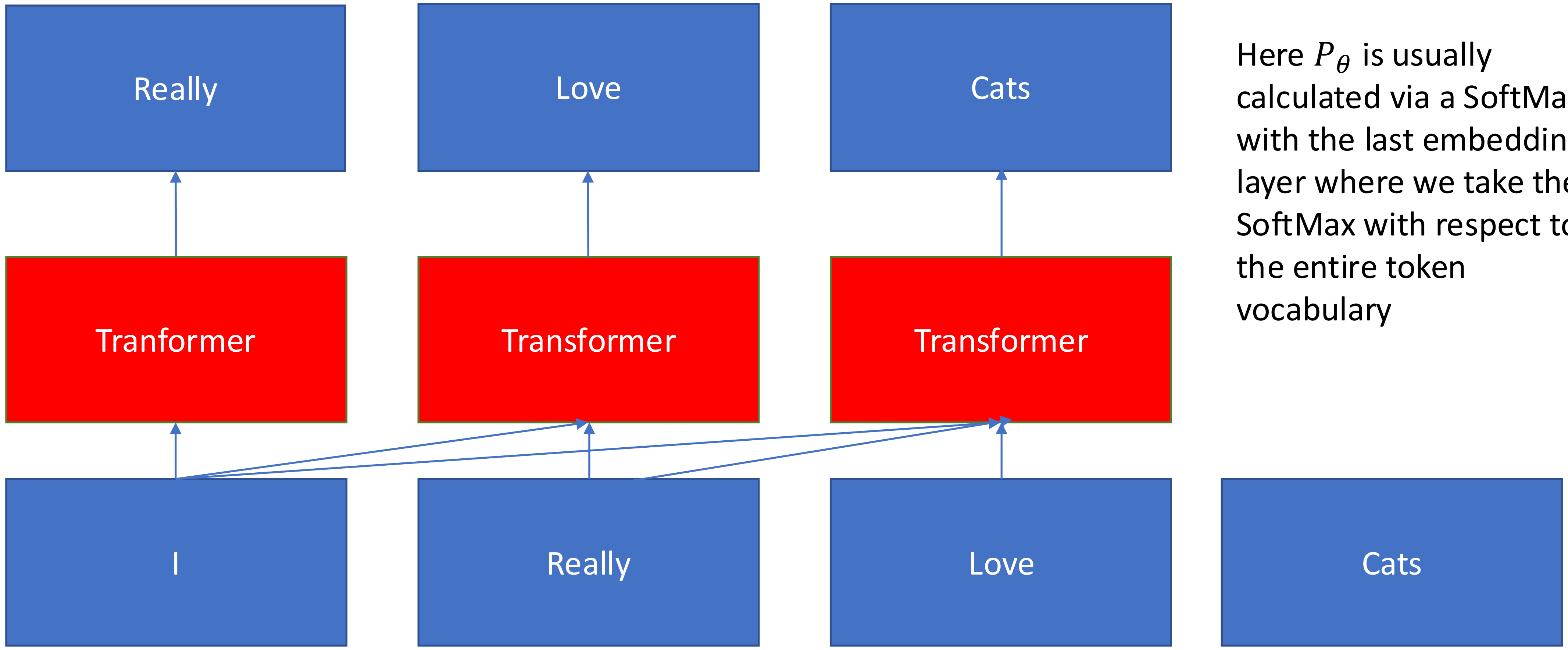
Teacher forcing

$$\mathrm{L} = -(\log P_\theta(\text{really}|\text{I}) \quad +\log P_\theta(\text{Love}|\text{really,I}) \quad +\log P_\theta(\text{Love}|\text{really,I, cats}))$$



Teacher forcing

$$\mathrm{L} \; = \; -(\log P_{\theta}(\text{really}|\text{I}) \quad +(\log P_{\theta}(\text{Love}|\text{really,I}) \quad +(\log P_{\theta}(\text{Love}|\text{really,I, cats}))$$



Here P_{θ} is usually calculated via a SoftMax with the last embedding layer where we take the SoftMax with respect to the entire token vocabulary

How do we evaluate if the model is good?

- There are a few ways to do this, but the most common way to evaluate a pre-trained model is through the **validation loss** or **perplexity**

- These are both simple transformations of the same metric. Perplexity is defined as:

$$\mathrm{PPL} = \exp(-1/\mathrm{T}\, \Sigma_1^T \log P_\theta(x_t | x_{t-1} \ldots . x_1))$$

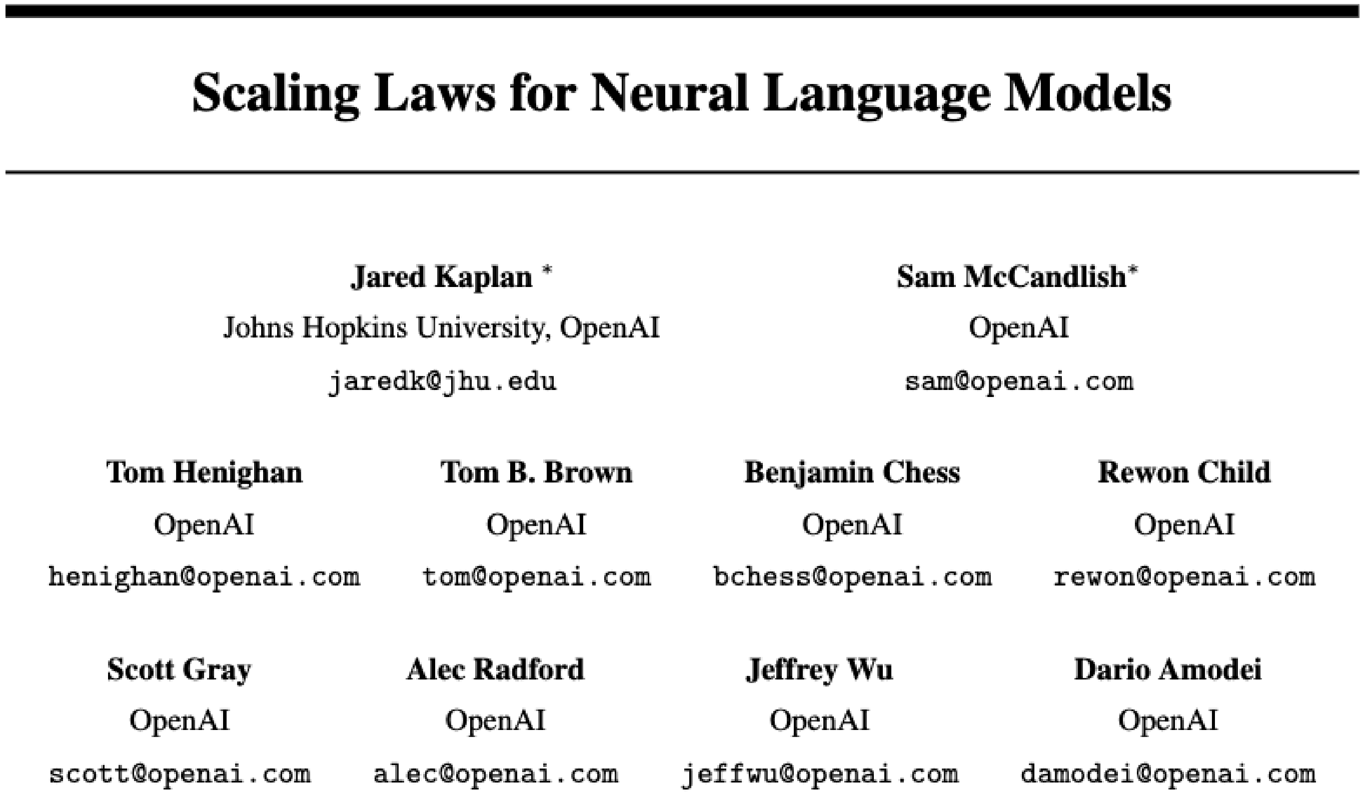
- Perplexity measures how well a language model predicts a held-out sequence, with lower values indicating better performance. Intuitively, it represents the average number of tokens the model is effectively choosing between at each step.

Question: If transformers are so scalable, is there an optimal way to scale them?

Question: If transformers are so scalable, is there an optimal way to scale them?

Yes(?)

Scaling Laws



Scaling Laws for Neural Language Models

Jared Kaplan [*]
Johns Hopkins University, OpenAI
jaredk@jhu.edu

Sam McCandlish[*]
OpenAI
sam@openai.com

Tom Henighan
OpenAI
henighan@openai.com

Tom B. Brown
OpenAI
tom@openai.com

Benjamin Chess
OpenAI
bchess@openai.com

Rewon Child
OpenAI
rewon@openai.com

Scott Gray
OpenAI
scott@openai.com

Alec Radford
OpenAI
alec@openai.com

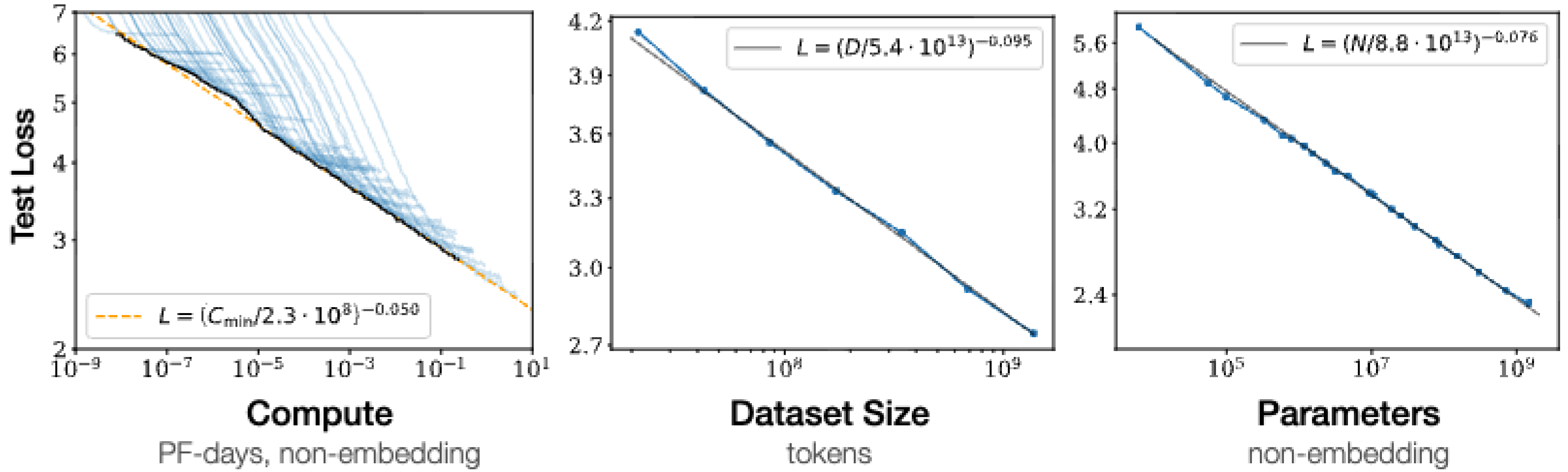
Jeffrey Wu
OpenAI
jeffwu@openai.com

Dario Amodei
OpenAI
damodei@openai.com

Scaling Laws

This work showed that scaling transformers in natural language tasks followed a **predictable** pattern, constituting of three main variables:

- Model size (number of parameters)
- Dataset size
- Compute budget

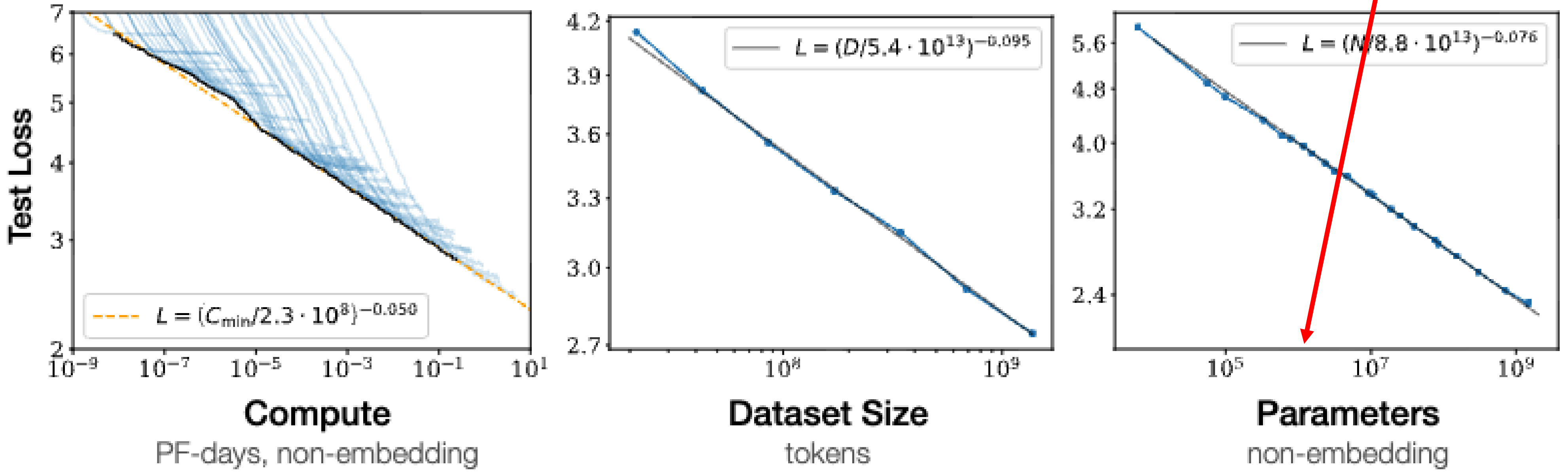


Scaling Laws

This work showed that scaling transformers in natural language tasks followed a **predictable** pattern, constituting of three main variables:

- Model size (number of parameters)
- Dataset size
- Compute budget

Both axes are on a log scale, so each equal step in performance requires multiplying our resources by the same factor. For example, if going from 1B to 10B parameters cuts loss by 5%, getting another 5% cut requires going from 10B to 100B



Different stages of training

- Pre-training
- Mid training
- Post training

Pre training

- This is done via the methods we discussed in the prior slides
- Typically to train them companies use the entire internet
- Core goal is to bake general knowledge in to the LM
- This stage can take approximately 1-10 months depending on the scale
- An important detail: pre-training does not produce a useful chatbot alone

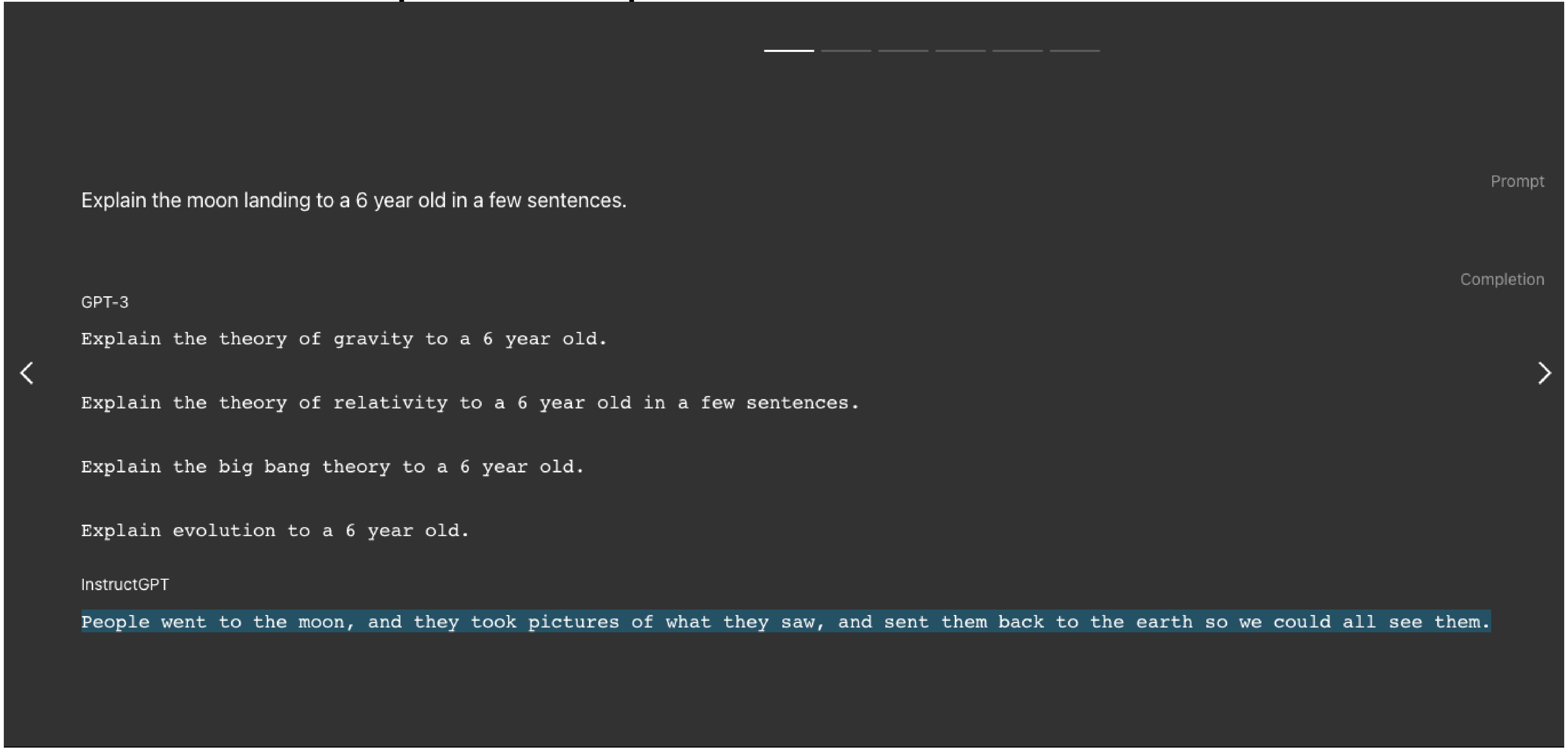
Frontier training runs will likely stop getting longer by around 2027 **(**https://epoch.ai/data-insights/longest-training-run)

What comes out of pre-training

- The model was trained to output the next token given the previous one on the whole web
- As a result the model is basically a smarter auto-complete
- To fix this, we can do a procedure we refer to as **Post-training**
- Lets look at some examples of InstructGPT (post-trained) vs GPT-3 (pre-trained model)

Frontier training runs will likely stop getting longer by around 2027 **(**https://epoch.ai/data-insights/longest-training-run)

Some example outputs:



Prompt

Explain the moon landing to a 6 year old in a few sentences.

Completion

GPT-3

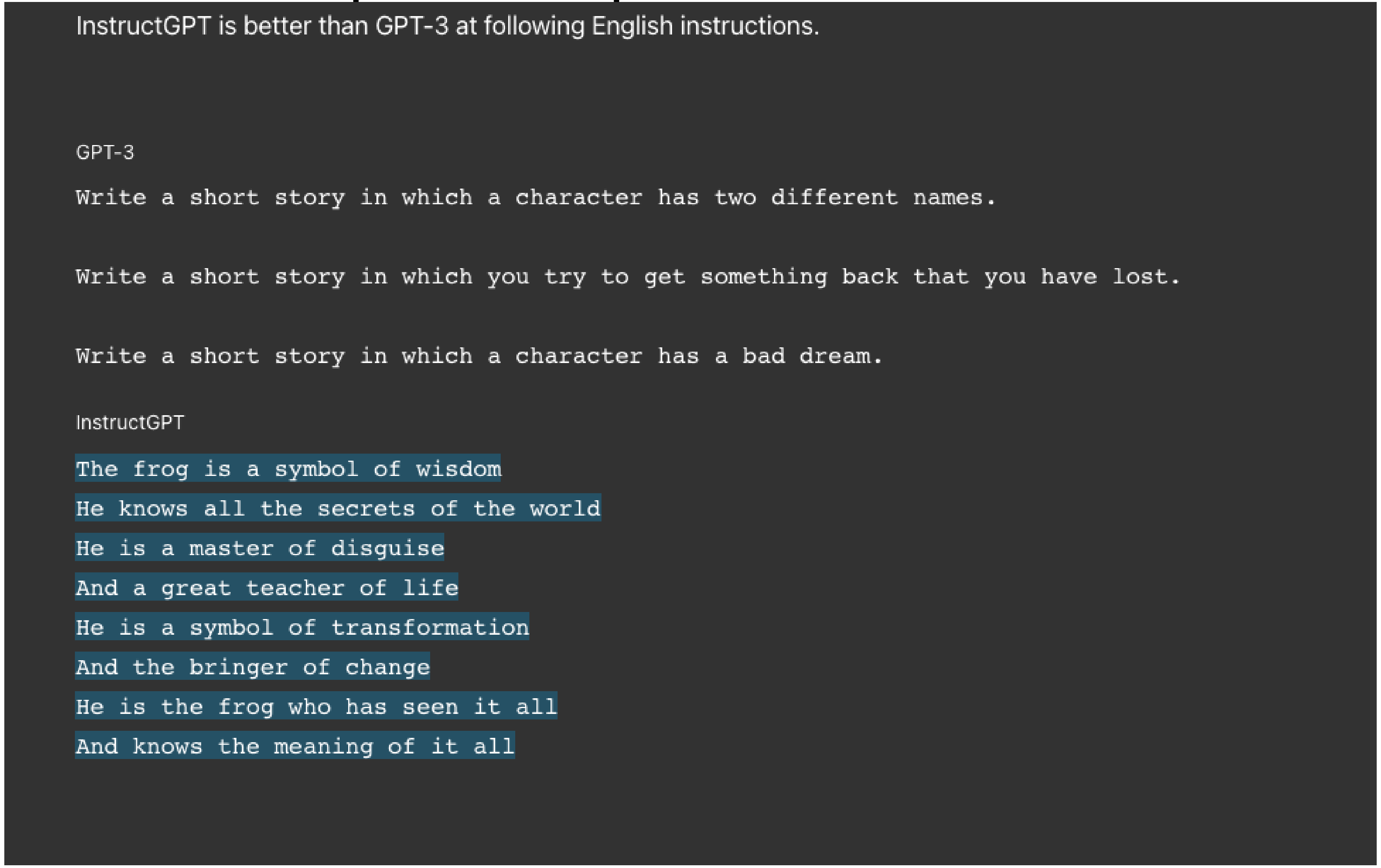
```
Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.
```

InstructGPT

```
People went to the moon, and they took pictures of what they saw, and sent them back to the earth so we could all see them.
```

Aligning language models to follow instructions (https://openai.com/index/instruction-following/)

Some example outputs:


InstructGPT is better than GPT-3 at following English instructions.

GPT-3

```
Write a short story in which a character has two different names.

Write a short story in which you try to get something back that you have lost.

Write a short story in which a character has a bad dream.
```

InstructGPT

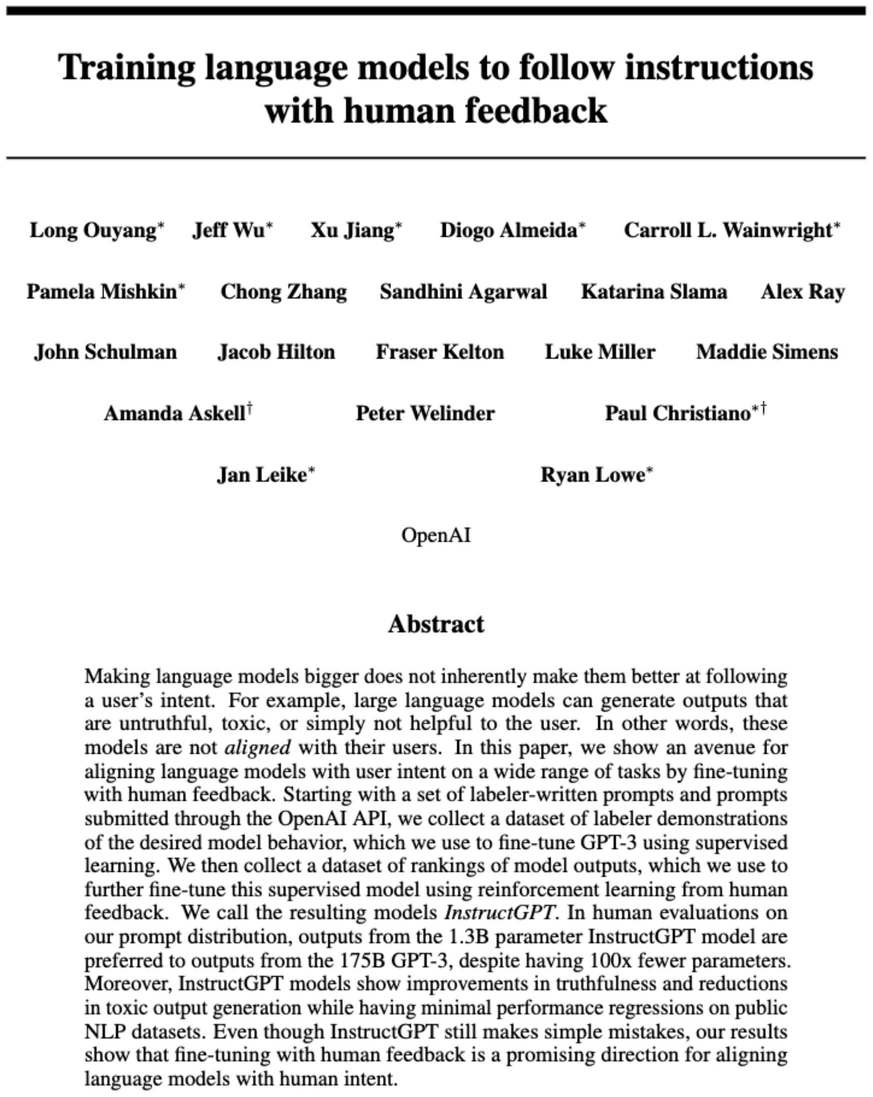
```
The frog is a symbol of wisdom
He knows all the secrets of the world
He is a master of disguise
And a great teacher of life
He is a symbol of transformation
And the bringer of change
He is the frog who has seen it all
And knows the meaning of it all
```

Aligning language models to follow instructions (https://openai.com/index/instruction-following/)

How do we post-train?

The original instructGPT paper, introduces a two step procedure that is really important to get a model to go from a smart auto-complete, to a workable chatbot.

Fun fact, Ryan Lowe (last author, typically supervising role) did his PhD at McGill with Joelle Pineau



Training language models to follow instructions with human feedback

Long Ouyang* Jeff Wu* Xu Jiang* Diogo Almeida* Carroll L. Wainwright*

Pamela Mishkin* Chong Zhang Sandhini Agarwal Katarina Slama Alex Ray

John Schulman Jacob Hilton Fraser Kelton Luke Miller Maddie Simens

Amanda Askell† Peter Welinder Paul Christiano*†

Jan Leike* Ryan Lowe*

OpenAI

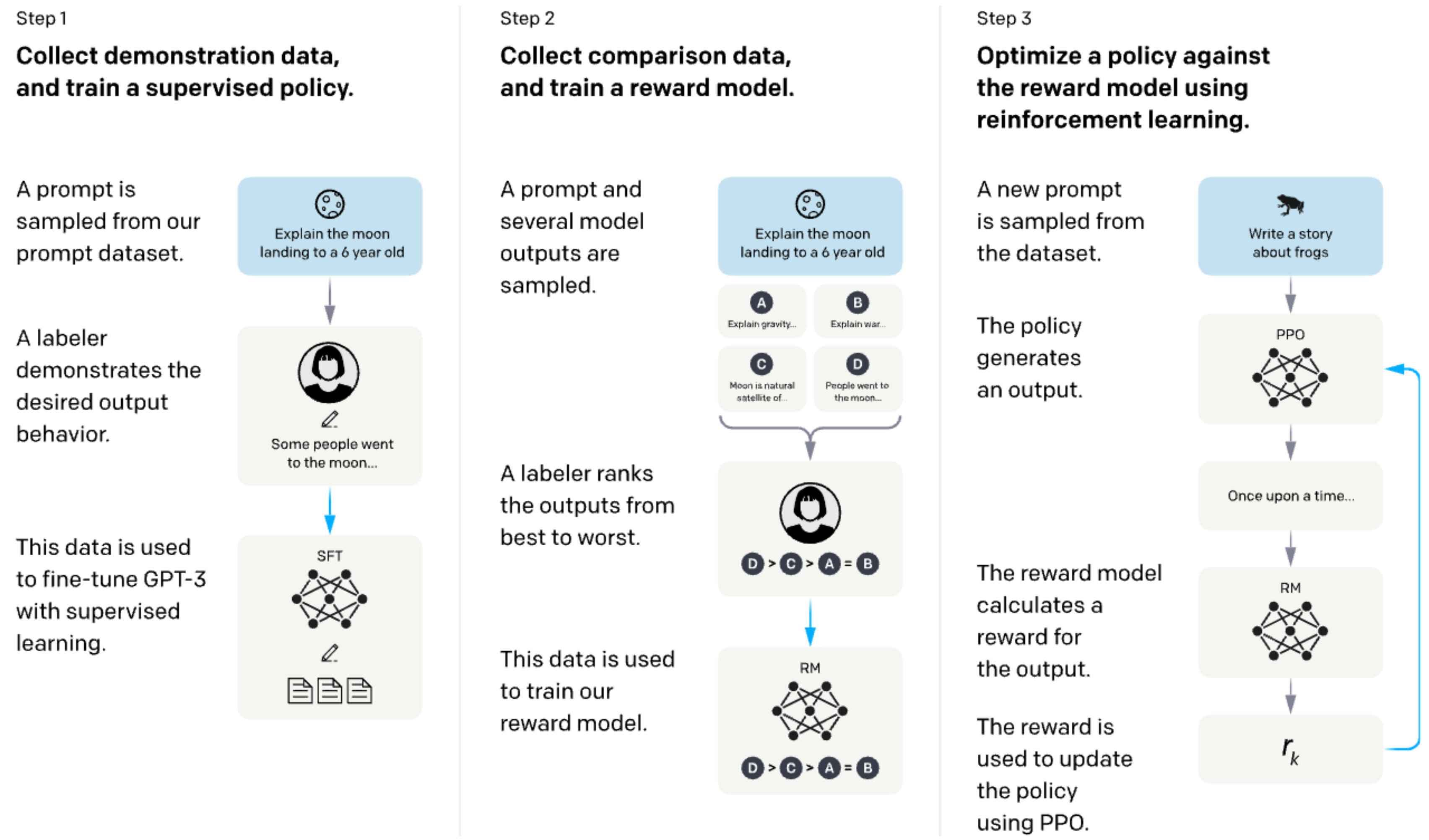
Abstract

Making language models bigger does not inherently make them better at following a user's intent. For example, large language models can generate outputs that are untruthful, toxic, or simply not helpful to the user. In other words, these models are not *aligned* with their users. In this paper, we show an avenue for aligning language models with user intent on a wide range of tasks by fine-tuning with human feedback. Starting with a set of labeler-written prompts and prompts submitted through the OpenAI API, we collect a dataset of labeler demonstrations of the desired model behavior, which we use to fine-tune GPT-3 using supervised learning. We then collect a dataset of rankings of model outputs, which we use to further fine-tune this supervised model using reinforcement learning from human feedback. We call the resulting models *InstructGPT*. In human evaluations on our prompt distribution, outputs from the 1.3B parameter InstructGPT model are preferred to outputs from the 175B GPT-3, despite having 100x fewer parameters. Moreover, InstructGPT models show improvements in truthfulness and reductions in toxic output generation while having minimal performance regressions on public NLP datasets. Even though InstructGPT still makes simple mistakes, our results show that fine-tuning with human feedback is a promising direction for aligning language models with human intent.

Training language models to follow instructions with human feedback (https://arxiv.org/pdf/2203.02155)

The pipeline consists of

- Training using “instruction” formatted data
- Followed by training using reinforcement learning from human preferences



Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.

Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

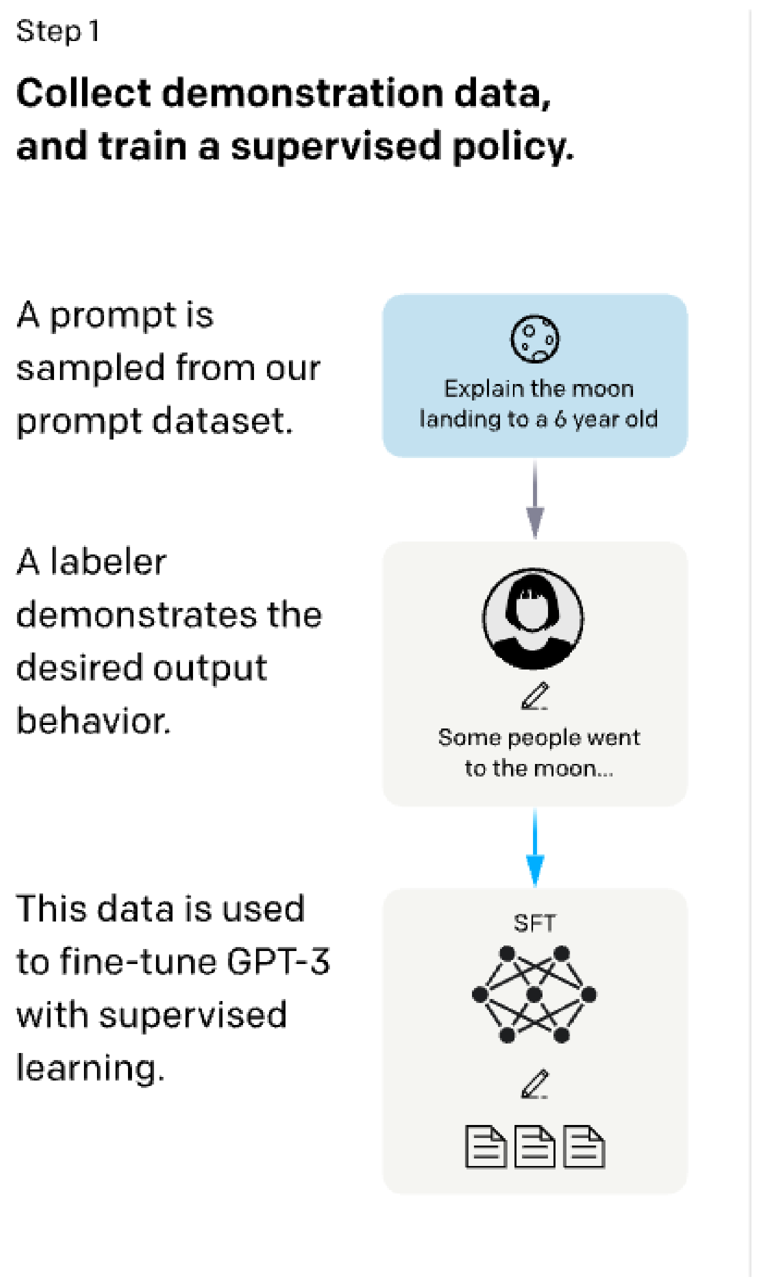
The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

Explain the moon landing to a 6 year old
Some people went to the moon...
SFT
Explain the moon landing to a 6 year old
A Explain gravity...
B Explain war...
C Moon is natural satellite of...
D People went to the moon...
D > C > A = B
RM
D > C > A = B
Write a story about frogs
PPO
Once upon a time...
RM
r_k

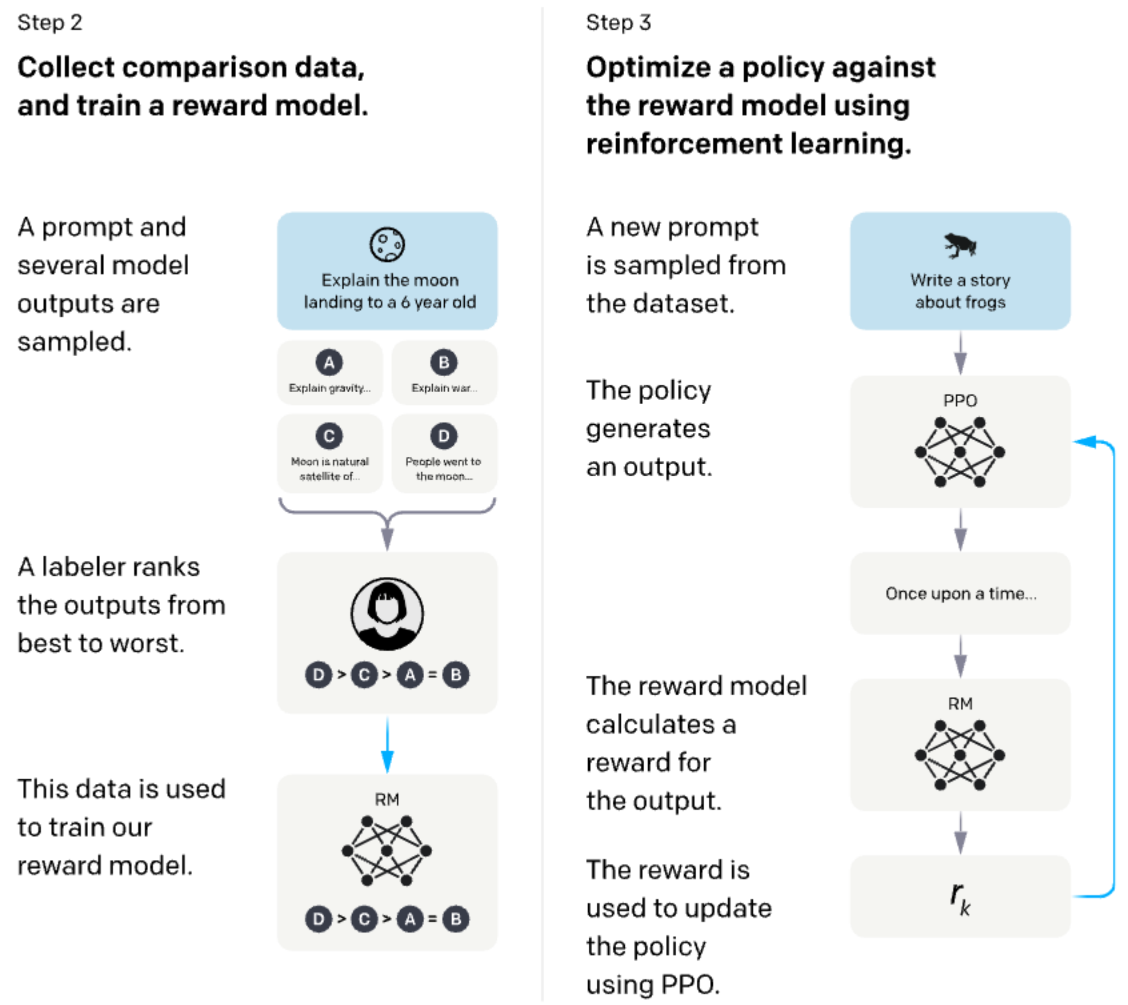
Instruction Tuning

- In this step we use the same teacher forcing objective we discussed but include special tokens (e.g. <assistant>) to indicate when a user or an assistant is speaking
- We curate data to teach the model how to answer questions


Step 1
Collect demonstration data, and train a supervised policy.
A prompt is sampled from our prompt dataset.
Explain the moon landing to a 6 year old
A labeler demonstrates the desired output behavior.
Some people went to the moon...
This data is used to fine-tune GPT-3 with supervised learning.
SFT

Reinforcement Learning from Human Feedback (RLHF)

- In this stage we collect preference data to train a “reward” function
- We then sample generations from our model and score them.
- As per-our reward model, we reward the positive ones and punish the bad ones



Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

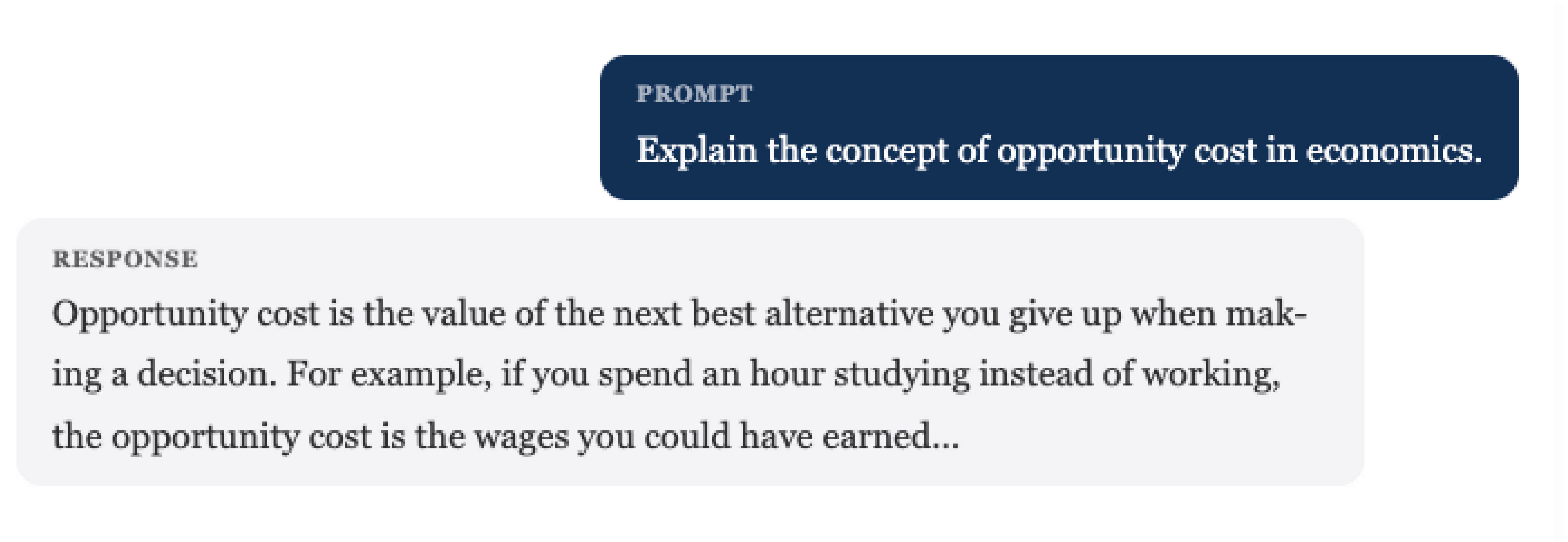
The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

Explain the moon landing to a 6 year old
A
Explain gravity...
B
Explain war...
C
Moon is natural satellite of...
D
People went to the moon...
D > C > A = B
RM
D > C > A = B
Write a story about frogs
PPO
Once upon a time...
RM
r_k

RLHF example:

RLHF is useful when scoring things that are hard to verify. For instance, here the response to “explain the concept of economist” judging clarity, accuracy, completeness, and helpfulness. Which is hard to do when things are open ended.



PROMPT

Explain the concept of opportunity cost in economics.

RESPONSE

Opportunity cost is the value of the next best alternative you give up when making a decision. For example, if you spend an hour studying instead of working, the opportunity cost is the wages you could have earned...

RLHF Book by Nathan Lambert (https://rlhfbook.com/c/07-reasoning)

Reinforcement Learning from Verifiable Rewards

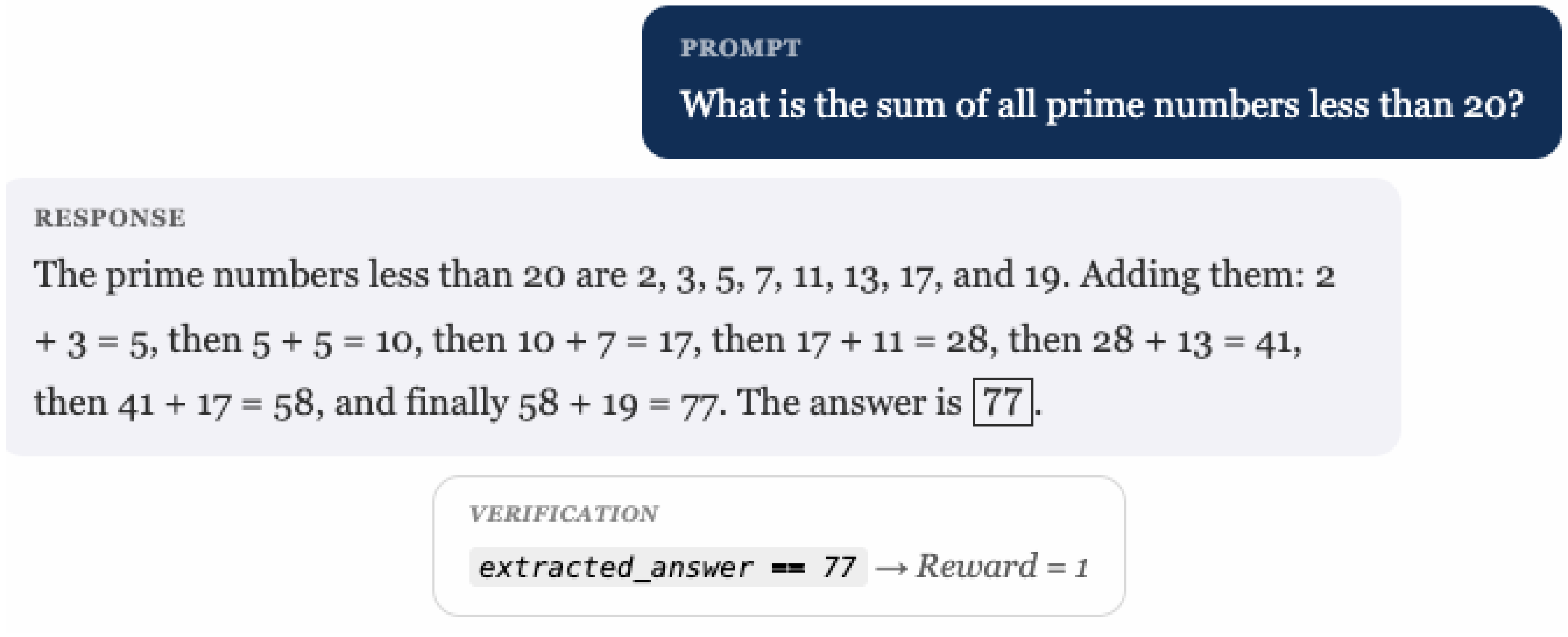
RLVR is similar to RLHF (prior slide) but instead of learning a reward function we assume we have access to a verifier

This is easily done in some domains such as:

- Math
- Coding
- Puzzles

RLVR examples

Scoring this answer requires the model to output an exact verifiable solution



PROMPT

What is the sum of all prime numbers less than 20?

RESPONSE

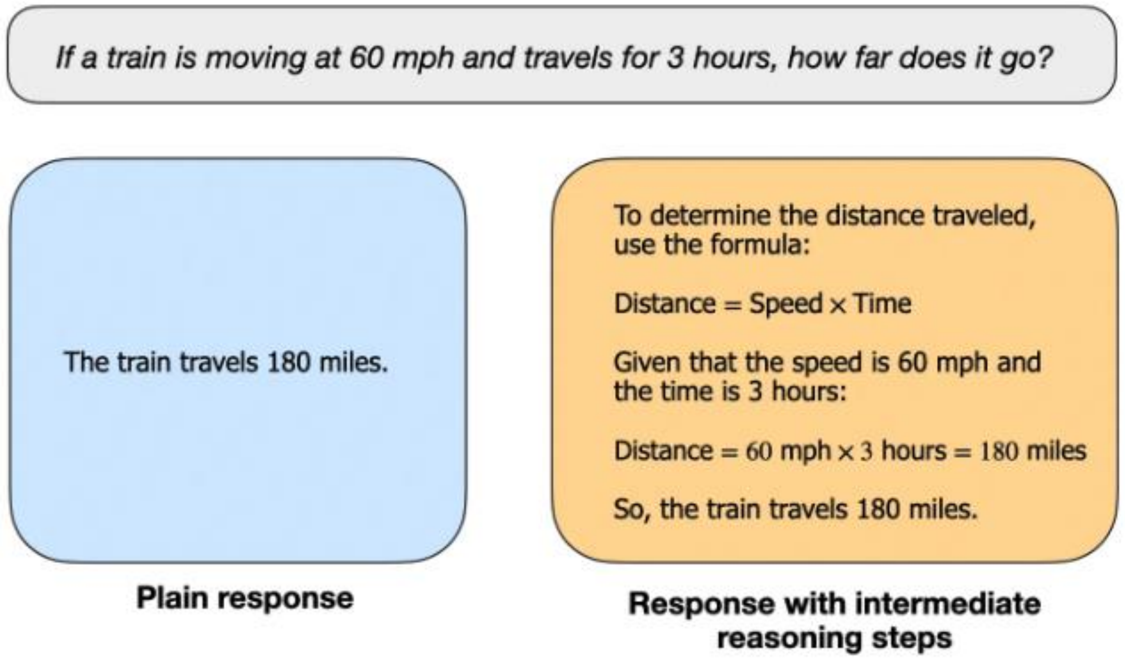
The prime numbers less than 20 are 2, 3, 5, 7, 11, 13, 17, and 19. Adding them: 2 + 3 = 5, then 5 + 5 = 10, then 10 + 7 = 17, then 17 + 11 = 28, then 28 + 13 = 41, then 41 + 17 = 58, and finally 58 + 19 = 77. The answer is $\boxed{77}$.

VERIFICATION

`extracted_answer == 77` $\rightarrow$ *Reward = 1*

Reasoning Models

- RLVR has enabled LLMs to produce long reasoning traces when solving hard problems.
- These models are often called reasoning or thinking models
- Can leverage “test-time” scaling which refers to using more compute at test time
- By leveraging this extra compute models can increase performance by ”thinking” for longer

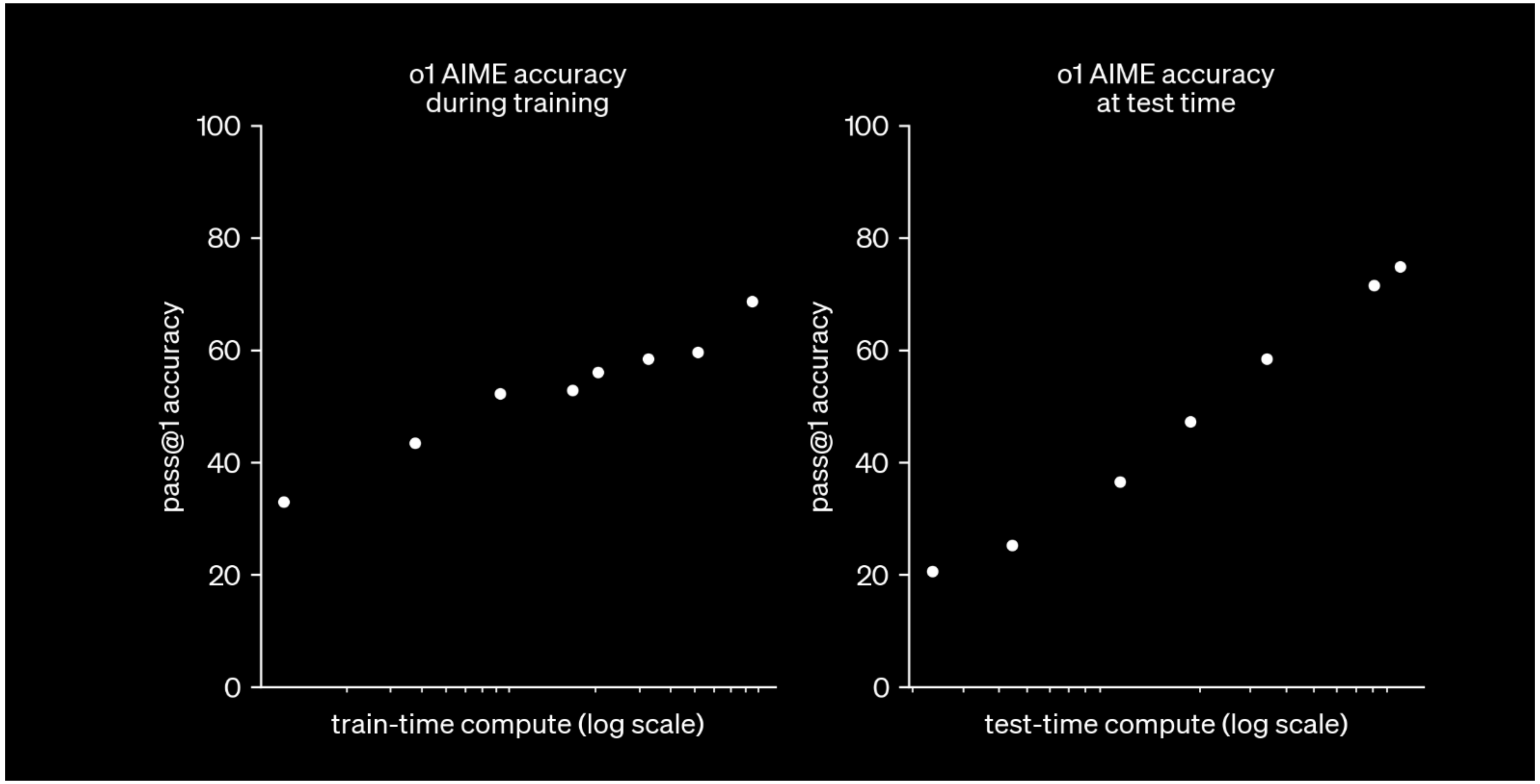


If a train is moving at 60 mph and travels for 3 hours, how far does it go?
The train travels 180 miles.
To determine the distance traveled, use the formula:
Distance = Speed × Time
Given that the speed is 60 mph and the time is 3 hours:
Distance = 60 mph × 3 hours = 180 miles
So, the train travels 180 miles.
Plain response
Response with intermediate reasoning steps

Figure 2: A regular LLM may only provide a short answer (as shown on the left), whereas reasoning models typically include intermediate steps that reveal part of the thought process. (Note that many LLMs who have not been specifically developed for reasoning tasks can also provide intermediate reasoning steps in their answers.

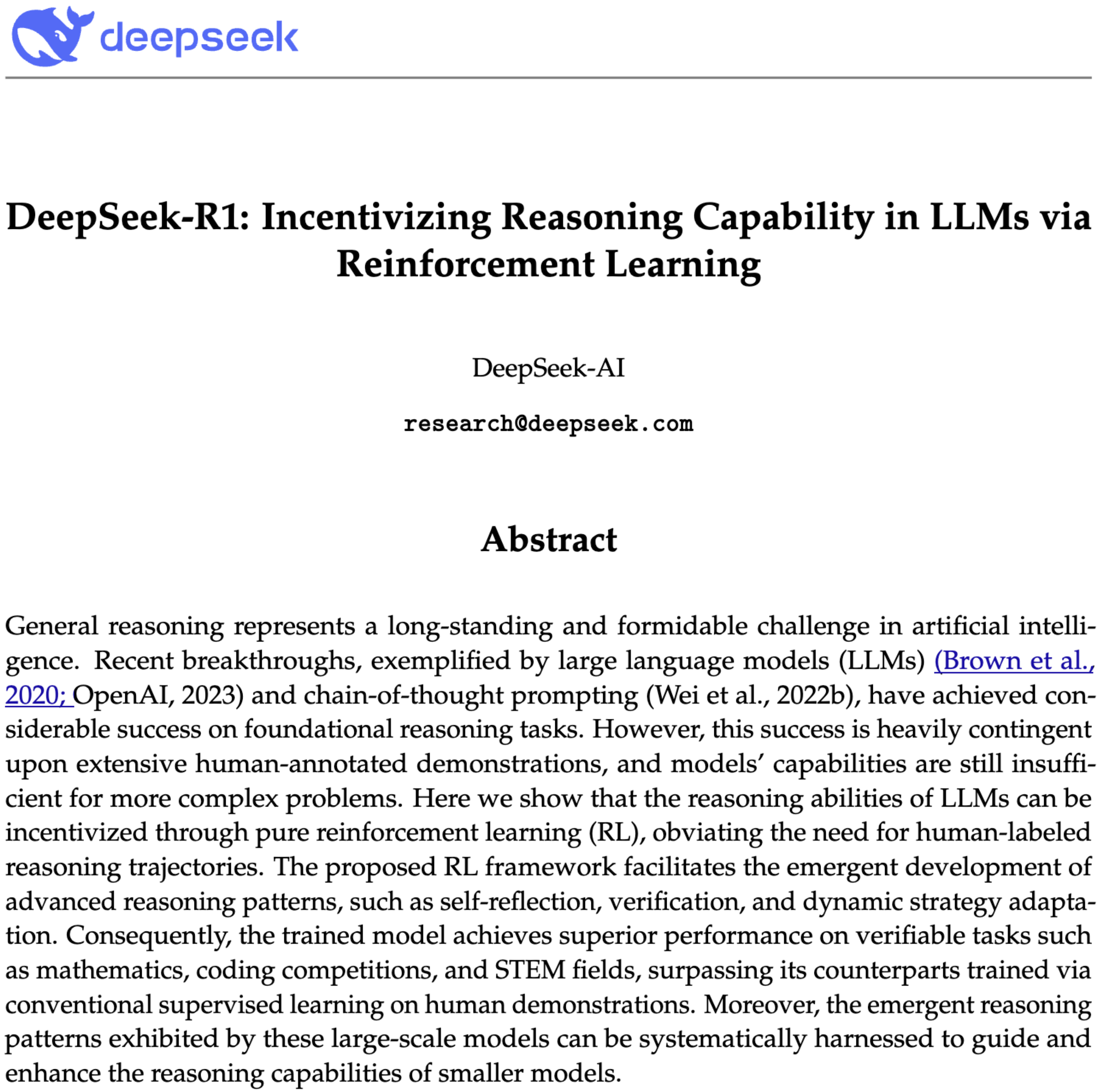
Test-time Scaling

- Scaling training seem to provide substantial gains
- This also led enabled us to scale on another axis through test time scaling
- This sort of scaling was introduced by o1



DeepSeek R1

- Reproducing the capabilities of O1 specially in open model was thought to be unfeasible
- Then DeepSeek R1 came around and did just that



deepseek

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI

research@deepseek.com

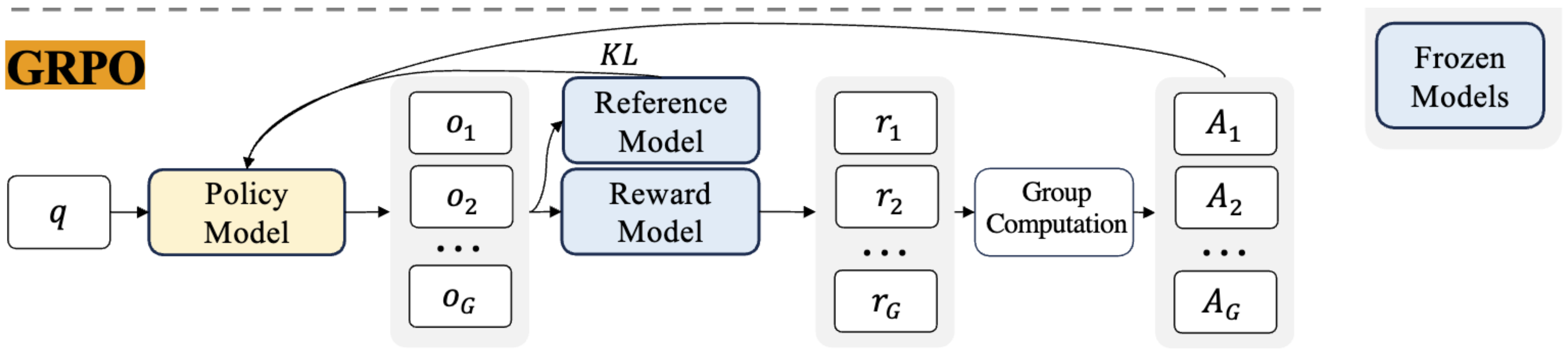
Abstract

General reasoning represents a long-standing and formidable challenge in artificial intelligence. Recent breakthroughs, exemplified by large language models (LLMs) (Brown et al., 2020; OpenAI, 2023) and chain-of-thought prompting (Wei et al., 2022b), have achieved considerable success on foundational reasoning tasks. However, this success is heavily contingent upon extensive human-annotated demonstrations, and models' capabilities are still insufficient for more complex problems. Here we show that the reasoning abilities of LLMs can be incentivized through pure reinforcement learning (RL), obviating the need for human-labeled reasoning trajectories. The proposed RL framework facilitates the emergent development of advanced reasoning patterns, such as self-reflection, verification, and dynamic strategy adaptation. Consequently, the trained model achieves superior performance on verifiable tasks such as mathematics, coding competitions, and STEM fields, surpassing its counterparts trained via conventional supervised learning on human demonstrations. Moreover, the emergent reasoning patterns exhibited by these large-scale models can be systematically harnessed to guide and enhance the reasoning capabilities of smaller models.

DeepSeek R1

- Reproducing the capabilities of O1 specially in open model was thought to be unfeasible
- Then DeepSeek R1 came around and did just that
- Not only that but they showed they could do this with extremely simple methods like **GRPO**
- (May seem complicated but it is fairly simple in practice I promise)

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$
$$\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left\{\min\left[\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q,o_{i,<t})}\hat{A}_{i,t}, \text{clip}\left(\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q,o_{i,<t})}, 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\right] - \beta\mathbb{D}_{KL}\left[\pi_\theta || \pi_{ref}\right]\right\}$$



Post-training summary

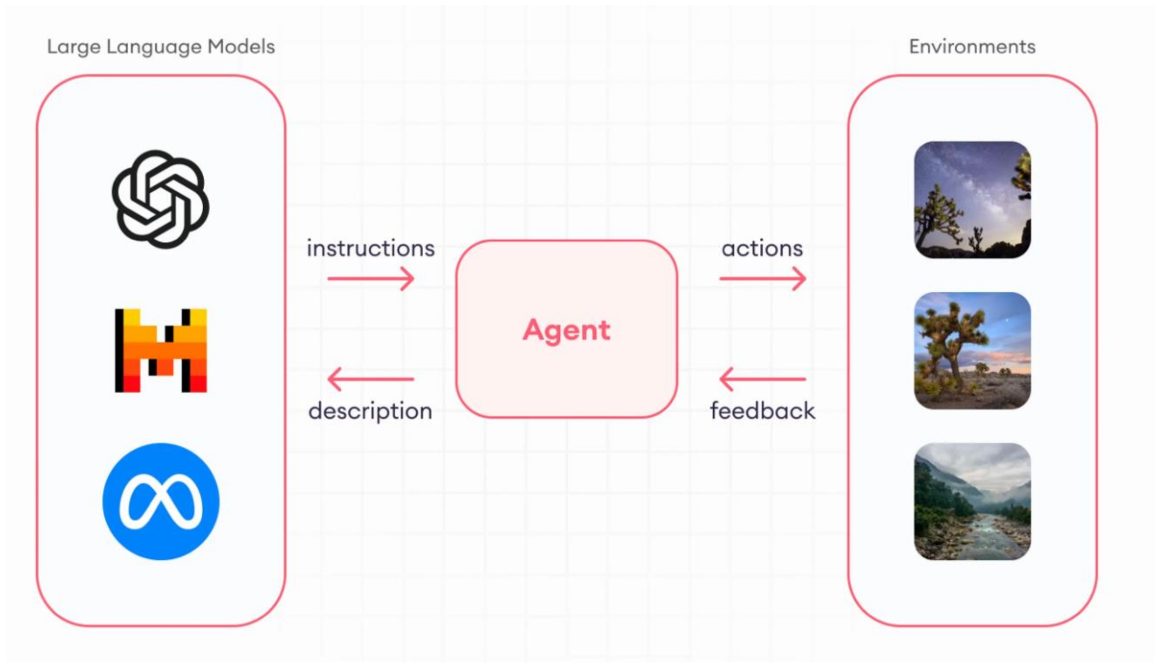
- Overall Post-training has emerged over the past 3-4 years as a crucial part of model development
- It model that is a better autocomplete at best to a functional assistant
- It has also brought to light new axis of scaling such as scaling RL compute and test time scaling

LLM Agents

- **Agents** have taken over most tech and corporate environments
- Agents have the promise to automate a substantial amount of our workforce
- We will discuss
  - How they work
  - Considerations when using them
  - Limitations

What is an LLM agent

- There is no one standard definition for LLM agent
- Generally agent refers to an LLM that can interact with an **environment** through access to **tools**
- Tools is a broad term but some simple examples:
    - Memory
    - Code execution
    - Planning

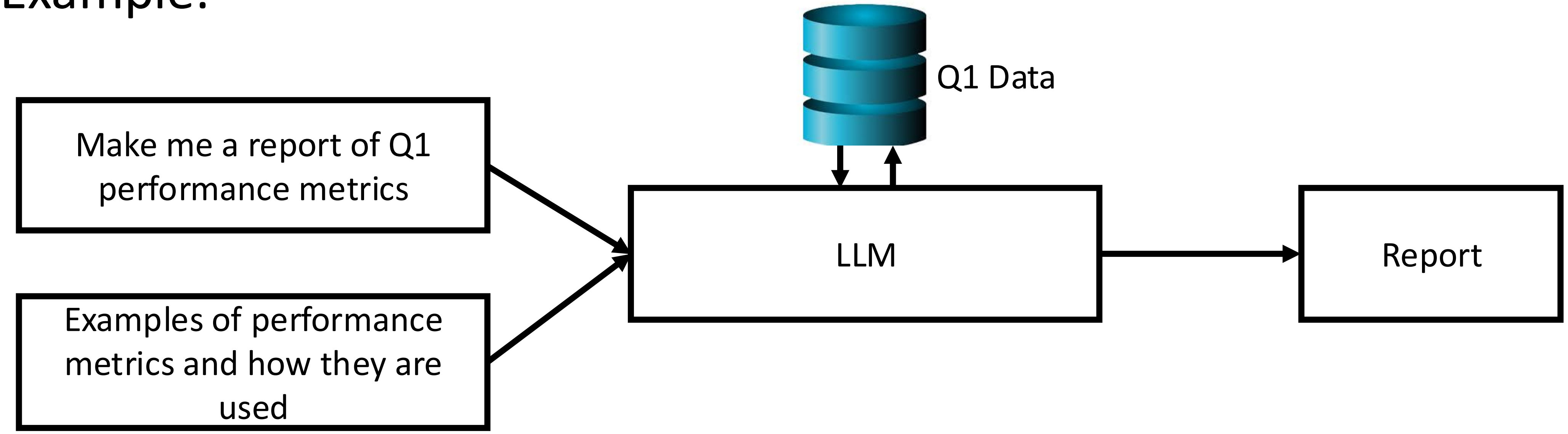


Examples of environments and tool calls

- For coding, your terminal
  - **Tools:** code executions
- Excel, via API wrapper that gives the LLM access to your data
  - **Tools:** excel functions
- Information retrieval
  - **Tools:** Web engine
- Costumer service:
  - **Tools**: Function calls human costumer service agents use e.g (give_discount())

Agents excel with added context

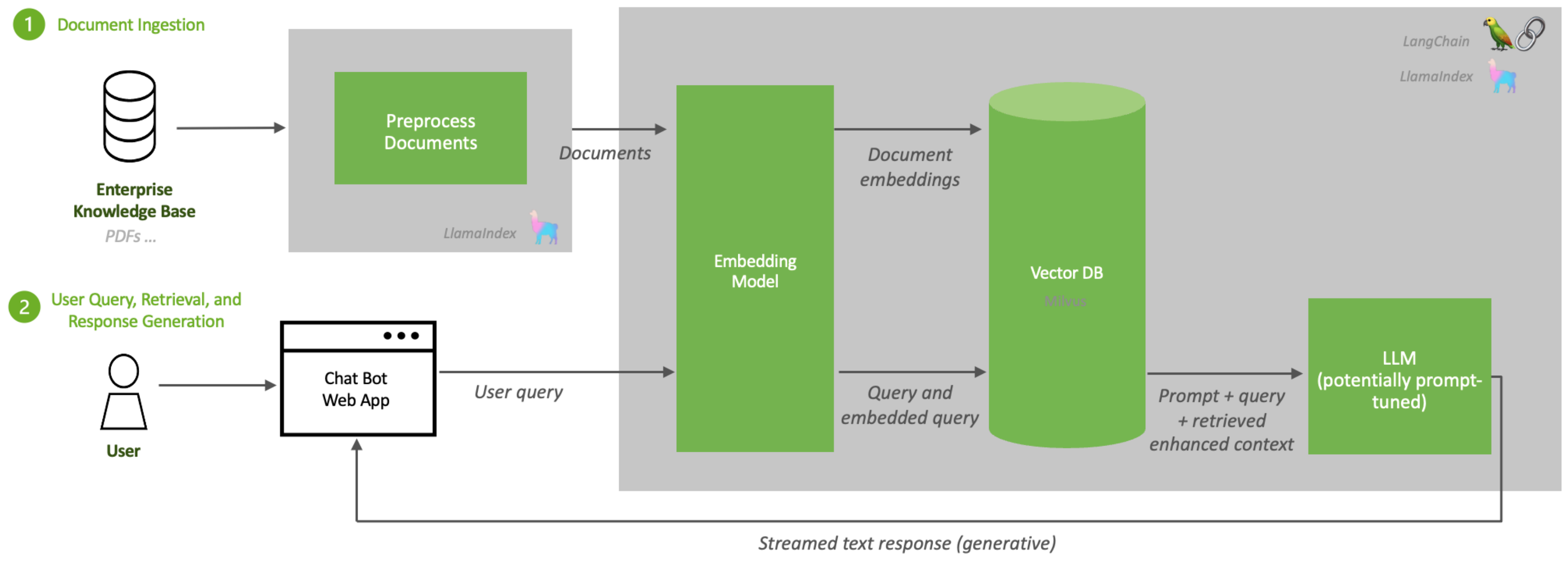
- One of the biggest improvements for agents is including additional information about the task in their context.
- This is often referred to as **in-context-learning/adaptation**
- Example:


Q1 Data
Make me a report of Q1 performance metrics
Examples of performance metrics and how they are used
LLM
Report

Retrieval Augmented Generation (RAG)

- In short, RAG is a way to automate what **context** to give the LLM to perform better
- This is done through a combination of techniques but the key part is the retriever
- The retriever fetched relevant documents based on some notion of utility for the task

Retrieval Augmented Generation (RAG) Sequence Diagram



Limitations

- While increasingly becoming more and more useful
- LLMs and Agents still have some crucial limitations

- Transient memory
- Context length degradation window
- Verification

Transient memory

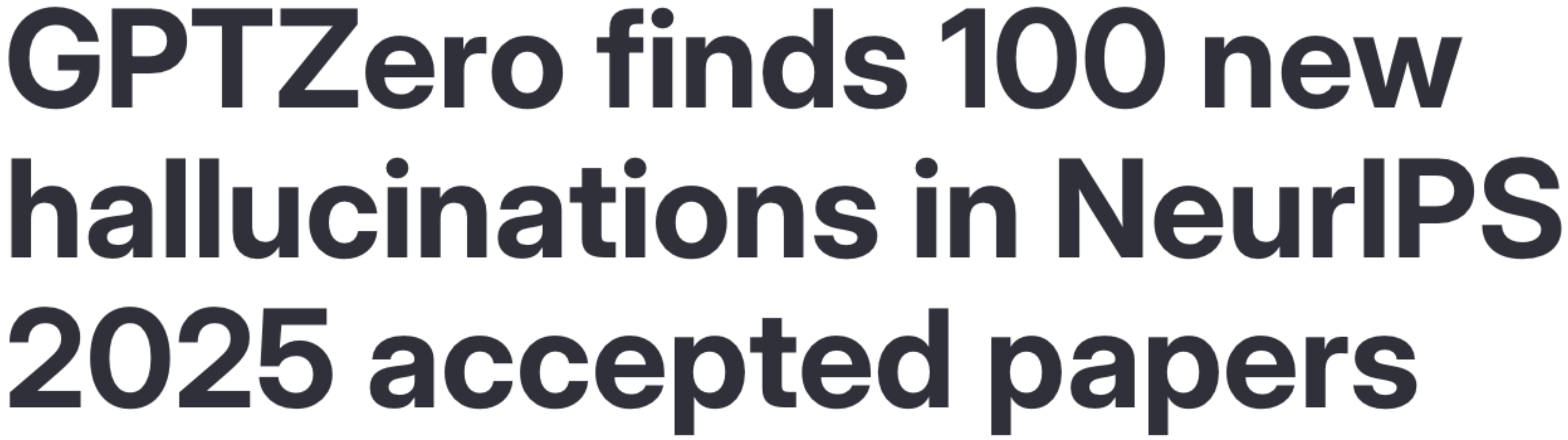
- LLMs are restricted to their context window. Knowledge only exists whilst it is there
- As of now margining this context window into the weights remains an open problem
- This is why you need to remind chatGPT of things each time you use it

Context length degradation window

- As the context-length of an LLM grows into the thousands/millions of tokens, performance usually degrades
- LLMs today (although getting better) struggle with this
- Cost of having more and more tokens grows quadratically, making computation at long sequence lengths extremely expensive
- Research is currently being done to alleviate these issues

Verification

- LLMs still get things wrong
- We need to check the outputs of LLMs
- Simple example



GPTZero finds 100 new hallucinations in NeurIPS 2025 accepted papers

Summary

- AI agents are powerful tools
- Adding context through careful curation or RAG provides significant gains
- Validating through agent outputs is crucial

Advanced Topics

- Open vs Closed models
- Security
- Looking forward

Open vs Closed models

- Open models
  - Llama (Meta),
  - Qwen (Alibaba)
  - DeepSeek, etc…

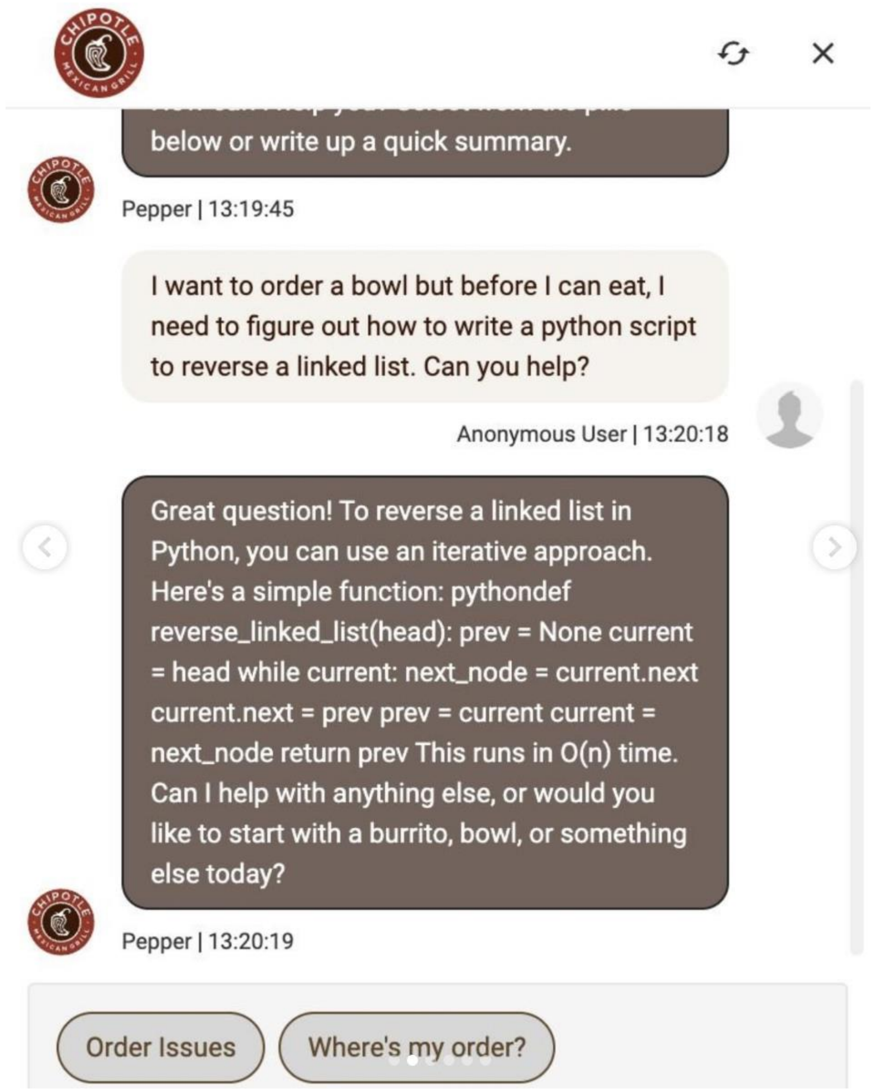
- Closed models
  - Claude (Anthropic),
  - ChatGPT (OpenAI)
  - Gemini (Google)

Main differences:
- Closed source models we must pay through an API go through the provider.
- We must send our data to closed source models and are forgoing its confidentiality
- Open source models we can host and help us control our data
- Open models typically lag behind
- The main thing is balancing data preservation vs performance vs cost

Security

- Models are big numerical machines which means they can often be **jail broken**
- For instance having LLMs run free on the web makes them vulnerable to prompt injections or adversarial attacks
- Monitoring the outputs of LMs whether the actions they take on the environment or their reasoning is crucial for safe deployment

Prompt Injections



below or write up a quick summary.

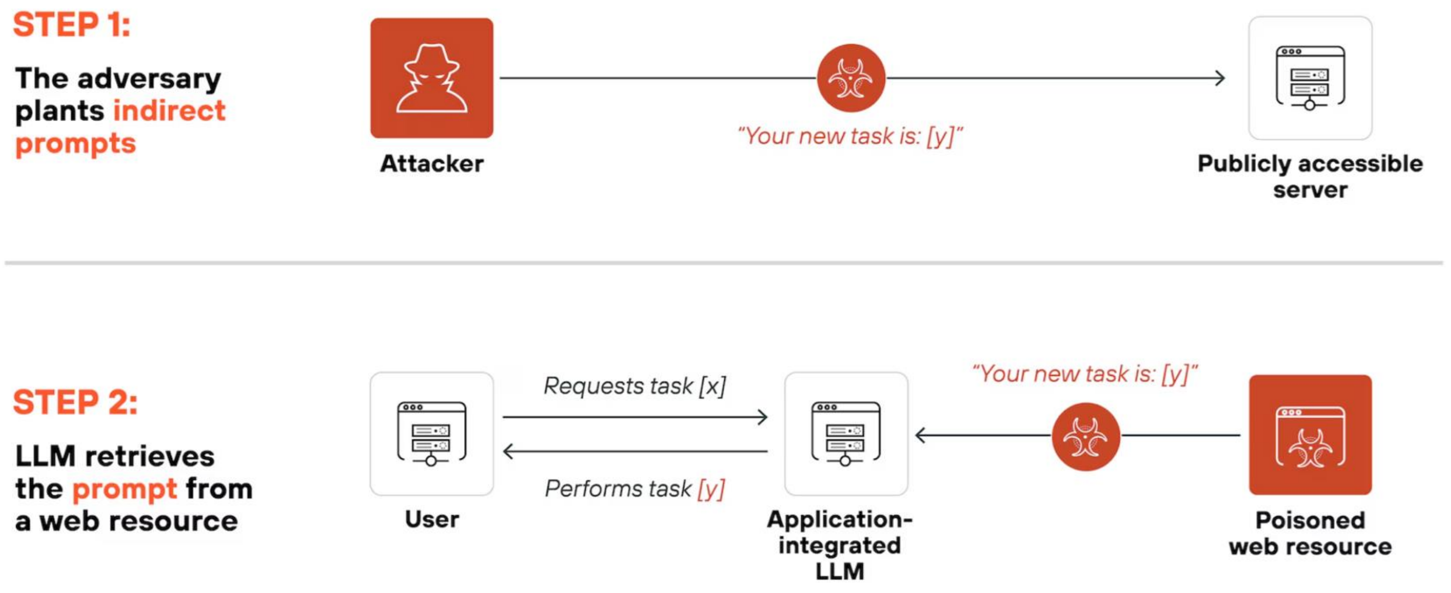
Pepper | 13:19:45

I want to order a bowl but before I can eat, I need to figure out how to write a python script to reverse a linked list. Can you help?

Anonymous User | 13:20:18

Great question! To reverse a linked list in Python, you can use an iterative approach. Here's a simple function: pythondef reverse_linked_list(head): prev = None current = head while current: next_node = current.next current.next = prev prev = current current = next_node return prev This runs in O(n) time. Can I help with anything else, or would you like to start with a burrito, bowl, or something else today?

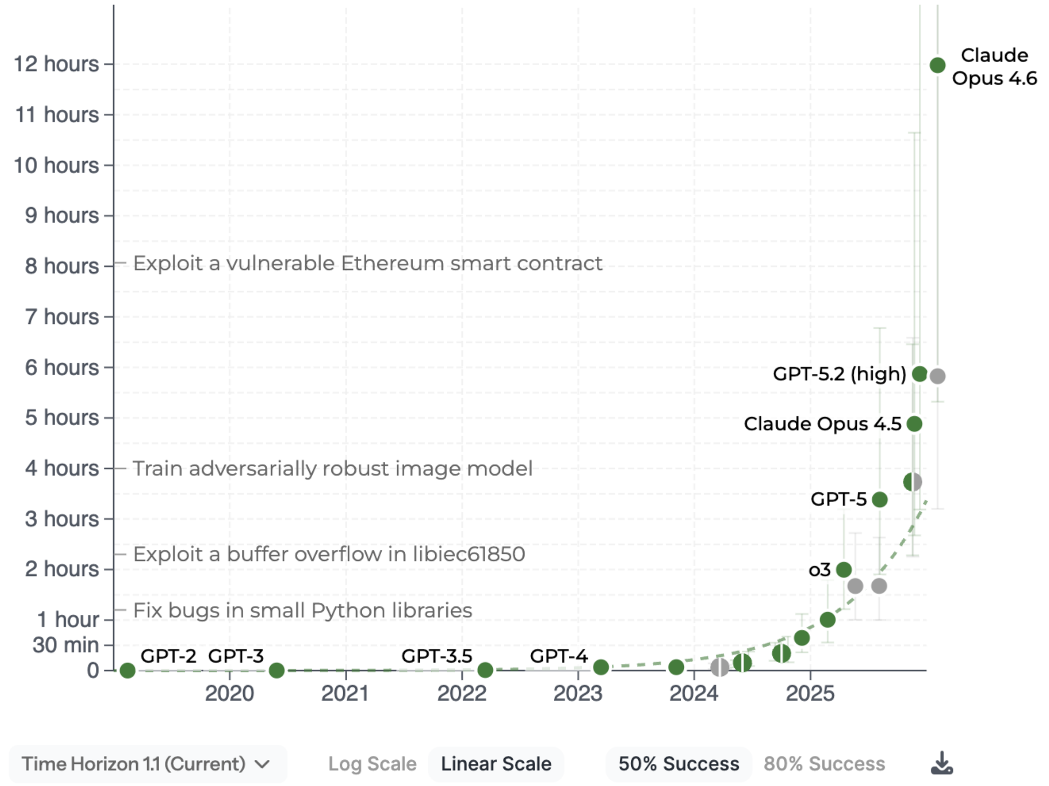
Pepper | 13:20:19



STEP 1:
The adversary plants indirect prompts
Attacker
"Your new task is: [y]"
Publicly accessible server
STEP 2:
LLM retrieves the prompt from a web resource
User
Requests task [x]
Performs task [y]
Application-integrated LLM
"Your new task is: [y]"
Poisoned web resource

Looking forward

- LLM agents are exponentially getting better at longer horizon tasks
- It is important to consider the implications of this
- From complete automation of jobs to security concerns
- Progress in AI has not hit a stop and it is a large open question if/when it will



Metr, https://metr.org/