Large Language Models:
from Transformer to ChatGPT

Guokan Shang

https://shang.tech/aboutme

updated on November 8, 2023

LaTeXof the slides: https://www.overleaf.com/read/rvwwvvwmxvyc

Chapter 1

Transformer

- Attention Is All You Need?[1]
- Yes!

Architecture

Embedding

Attention

1) Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

Architecture

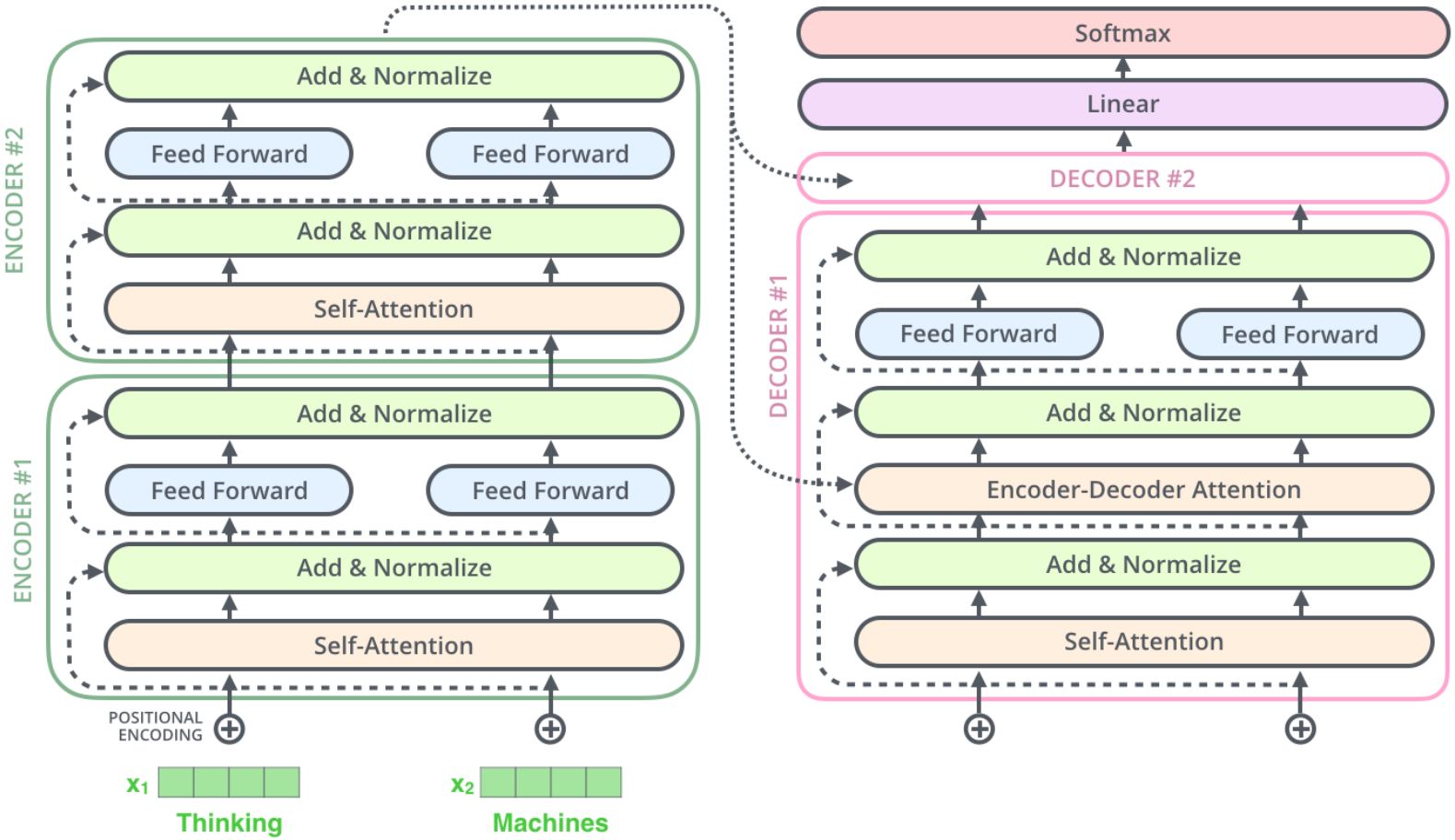


Figure: Transformer architecture.[1,2]

1) https://jalammar.github.io/illustrated-transformer/
2) http://nlp.seas.harvard.edu/annotated-transformer/

Byte-Pair Encoding

- Out-of-vocabulary, decomposition, Sinohydro ⇒ Sino, hydro

counts	
5	l o w _
2	l o w e s t _
6	n e w e r _
3	w i d e r _
2	n e w _

merge	current vocabulary
	_, d, e, i, l, n, o, r, s, t, w
(e, r)	_, d, e, i, l, n, o, r, s, t, w, er
(er, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_
(n, e)	_, d, e, i, l, n, o, r, s, t, w, er, er_ , ne
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_ , ne, new

Table: Corpus, merges, and vocabulary.[1] _ denotes end-of-word symbol.

- other subword tokenization algorithms: WordPiece, SentencePiece, Unigram.

1) `https://web.stanford.edu/~jurafsky/slp3/`

Positional Encoding

$$PE(pos, 2i) = sin(pos/10000^{2i/d_{model}})$$
$$PE(pos, 2i+1) = cos(pos/10000^{2i/d_{model}})$$

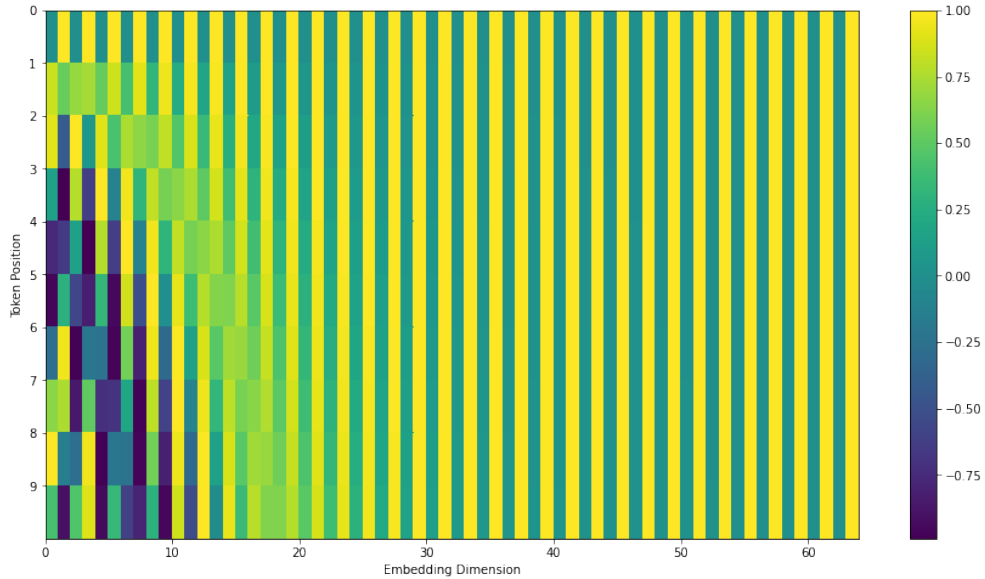


Figure: Sinusoid positional encoding.[1]

• RNN • Parallel computing • Learned positional embeddings

1) https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-

Attention

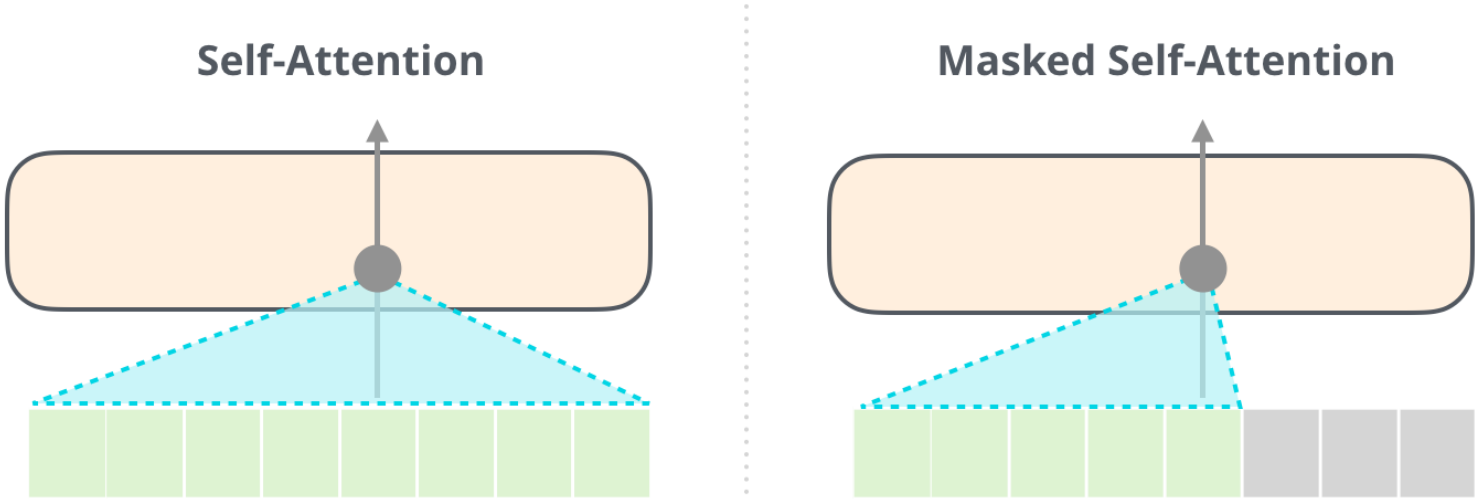


Figure: Self-attention of Encoder vs. Decoder.[1]

- Encoder-Decoder Attention (source-target alignment)

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

Query-Key-Value

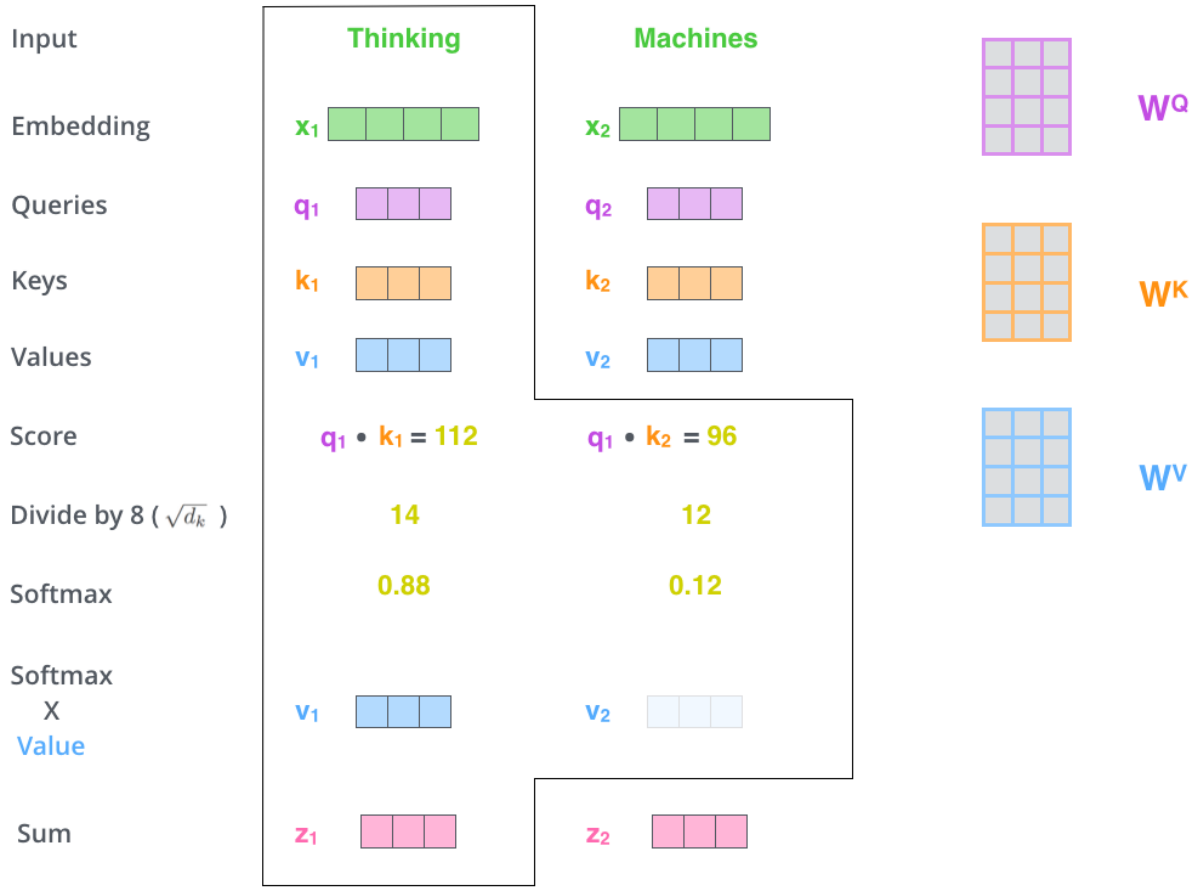


Figure: Self-attention calculation.

- Multi-Head Attention: $\{W_0^Q, W_0^K, W_0^V\}, \{W_1^Q, W_1^K, W_1^V\}, ...$

Attention

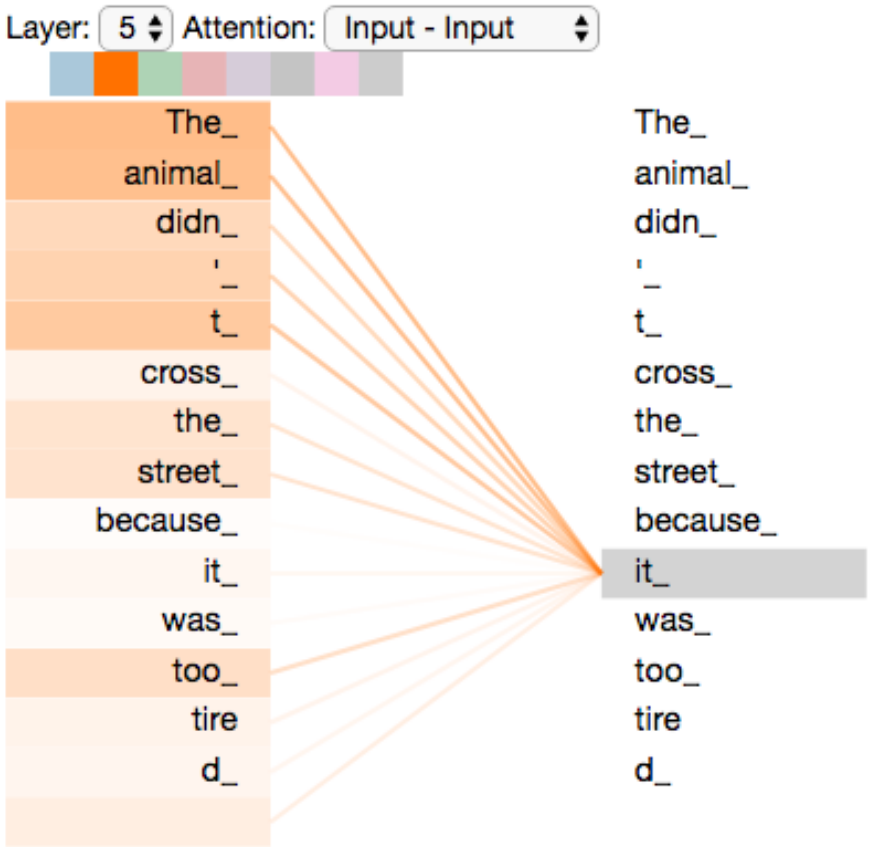


Figure: As we are encoding the word "it" in encoder #5 (the top encoder in the stack), part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it".

Decoding

- **Greedy search**:
  $w_t = \text{argmax}_w P(w|w_{1:t-1}) \Rightarrow$ *"The nice woman"*
- **Beam search**: keep the most likely *num_beams* of hypotheses at each time step. $\Rightarrow$ *"The dog has"*
  What if *num_beams* $=$ *num_vocab*? Viterbi?
- **Sampling**: $w_t \sim P(w|w_{1:t-1})$
  - $\frac{exp(y_i/T)}{\sum_{k=1}^{n} exp(y_k/T)}$
    low **temperature** softmax: [0.01,0.01,0.98] $\Rightarrow$ *controlled*
    high temperature softmax: [0.2,0.2,0.6] $\Rightarrow$ *open-ended*
  - **Top-K**: choose the top K tokens with the highest probabilities, remove the rest, and adjust their probabilities to sum to 1.
  - **Top-p** (nucleus) sampling: choose from the smallest possible set of words whose cumulative probability exceeds the probability p.

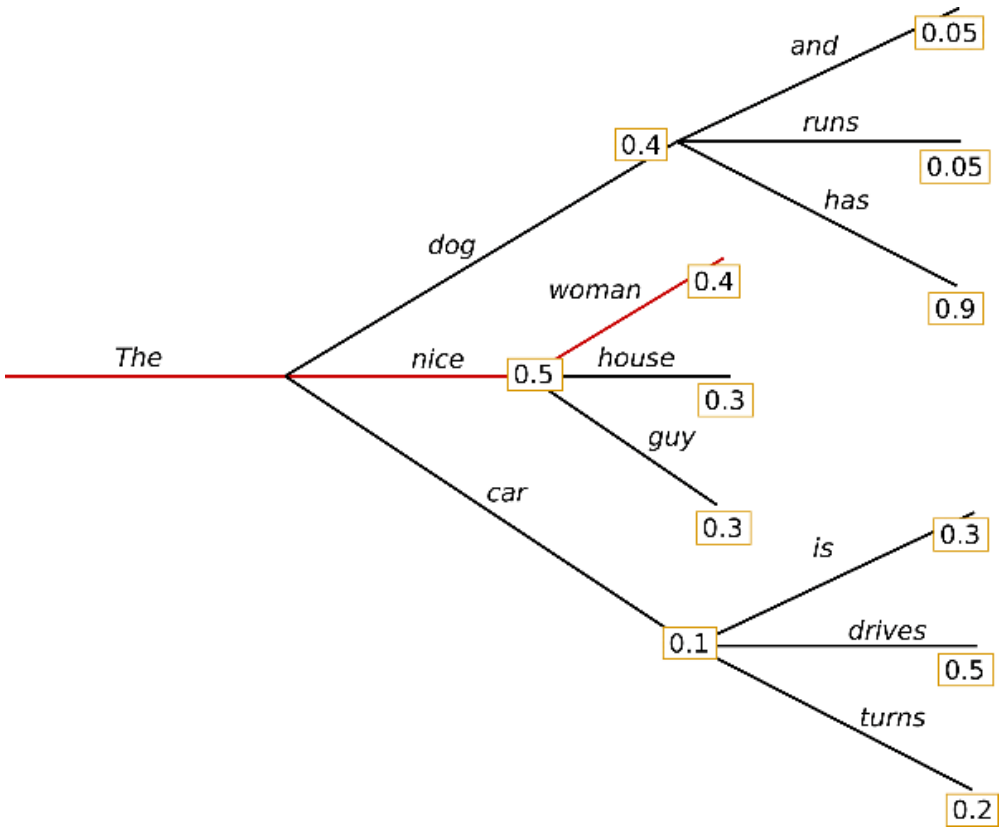


Figure: Decoding paths.

1) `https://huggingface.co/blog/how-to-generate`

Chapter 2

Pre-training (Large) Language Models

- Pre-training objectives: MLM, CLM, PLM, denoising
- General purpose: modeling language + knowledge

BERT | T5 | GPT

Language Model

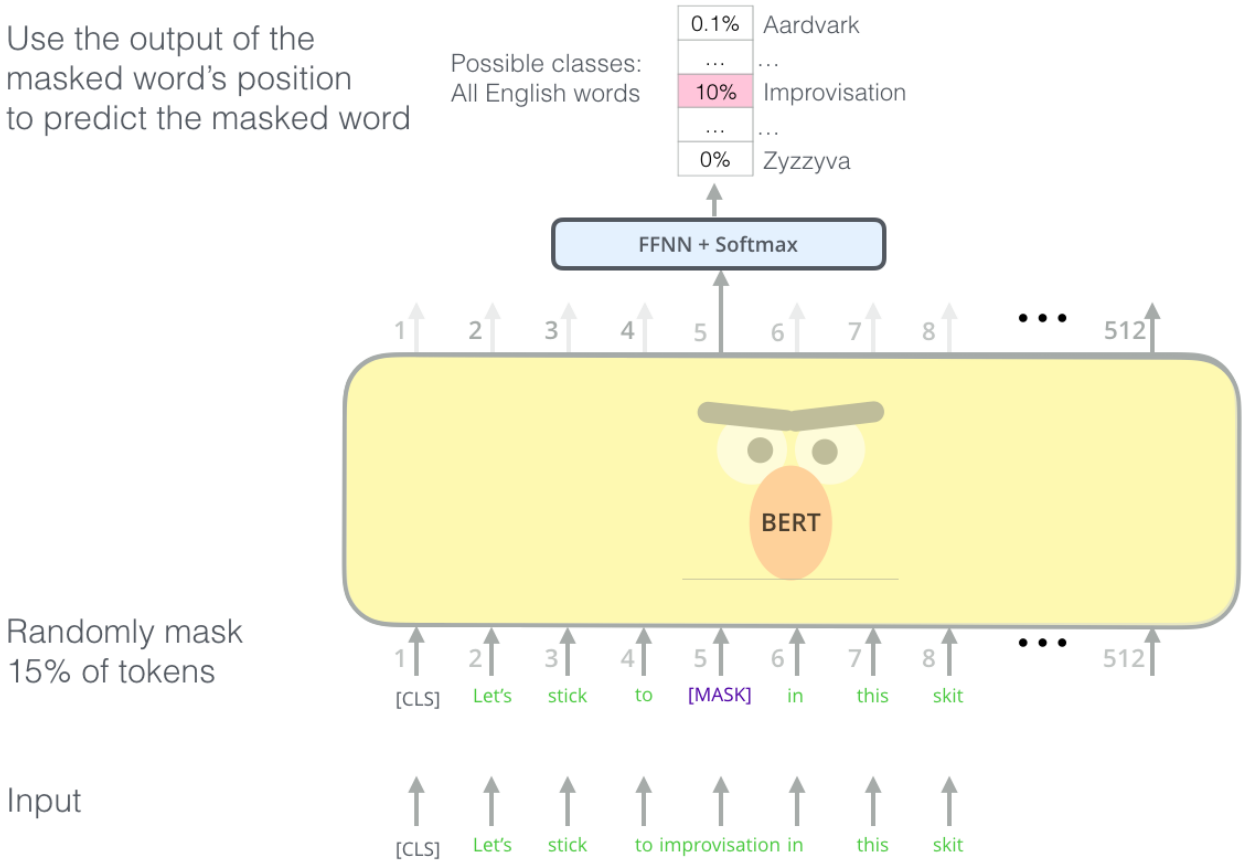
Models that assign probabilities to sequences of words are called **language models**[1].

- P(all of a sudden I notice three guys standing on the sidewalk) >
  P(on guys all I of notice sidewalk three a sudden standing the)
- P(?|Please turn your homework ?) $\Rightarrow$ in > over > > the > elephant
  `uni-directional`
  shorter context size, less accurate prediction: P(?|your homework ?)
- P(?|Please turn ? homework)
  `bi-directional`
- P(he briefed reporters on the main contents of the statement |
  il a informé les journalistes des principaux éléments de la déclaration) >
  P(he to reporters introduced main content |
  il a informé les journalistes des principaux éléments de la déclaration)
  `sequence-to-sequence`

1) `https://web.stanford.edu/~jurafsky/slp3/`

BERT | Encoder-only | MLM

Masked Language Modeling (MLM) predicts a masked token in a sequence, and the model can attend to tokens bidirectionally.[1] *Distributional hypothesis* of language?



• Contextualized word embeddings: Apple and apple.

1) `http://jalammar.github.io/illustrated-bert/`

Sentence-BERT[1]

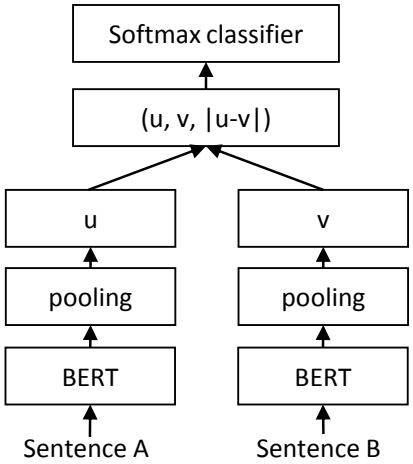


Figure: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

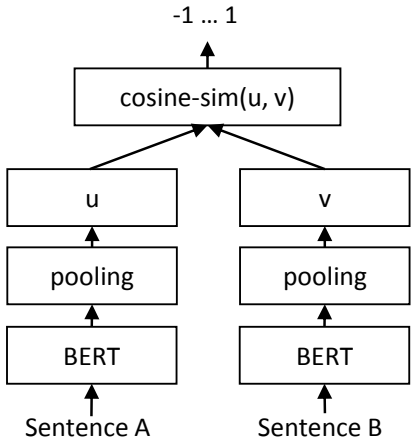


Figure: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

- Semantic textual similarity: `https://www.sbert.net`

```
model = SentenceTransformer("all-MiniLM-L6-v2")
emb1 = model.encode("This is a red cat with a hat.")
emb2 = model.encode("Have you seen my red cat?")
cos_sim = util.cos_sim(emb1, emb2)
```

- Keyword extraction: `https://github.com/MaartenGr/KeyBERT`

1) Reimers, Nils, and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks." arXiv preprint arXiv:1908.10084 (2019).

T5 | Encoder-decoder | MLM

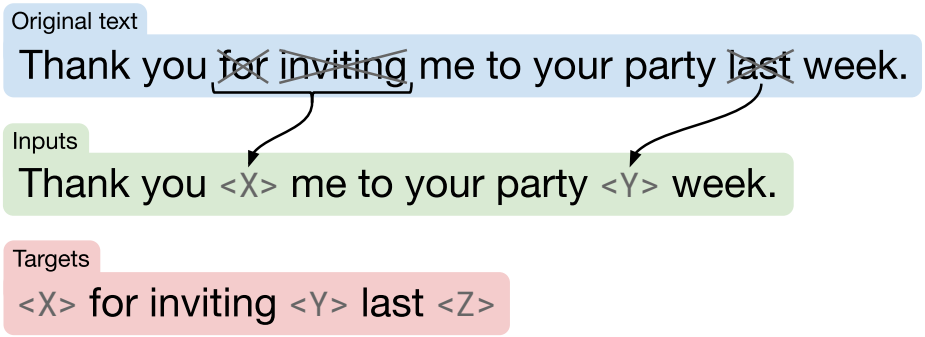


Figure: **PRE-TRAINING**: The words "for", "inviting" and "last" (marked with an $\times$) are randomly chosen for corruption. Each consecutive span of corrupted tokens is replaced by a sentinel token (shown as <X> and <Y>) that is unique over the example. The output sequence then consists of the dropped-out spans, delimited by the sentinel tokens used to replace them in the input plus a final sentinel token <Z>.

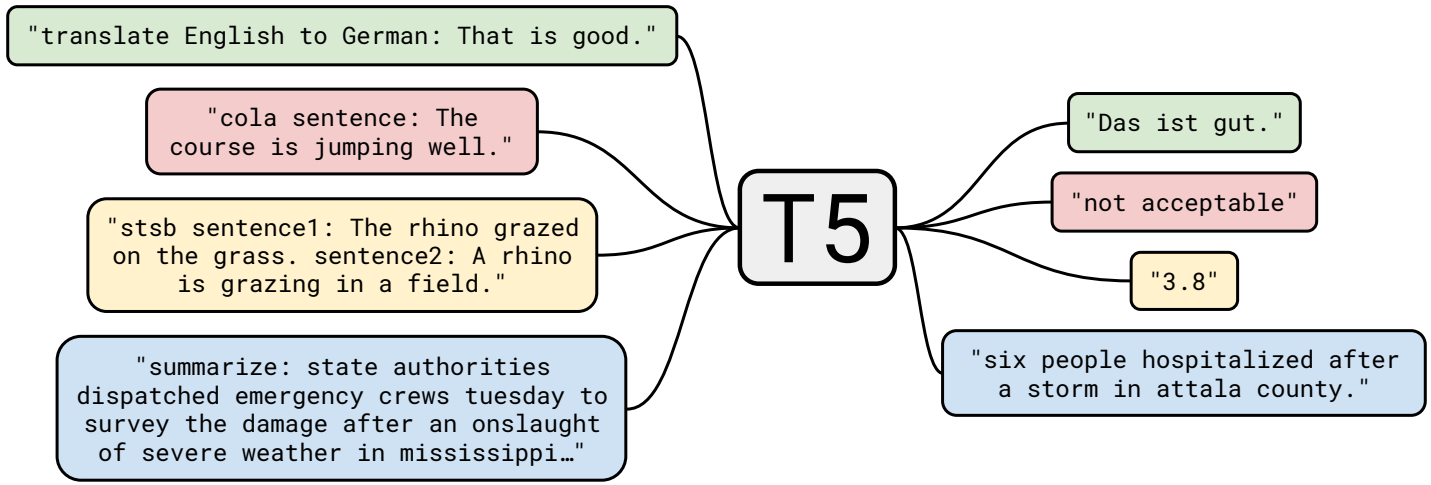


Figure: **INSTRUCTION-TUNING**: every task is cast as feeding text as input to generate some target text.[1]

1) Raffel, Colin, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." The Journal of

GPT | Decoder-only | CLM

Causal language modeling (CLM) predicts the next token in a sequence of tokens, and the model can only attend to tokens on the left.

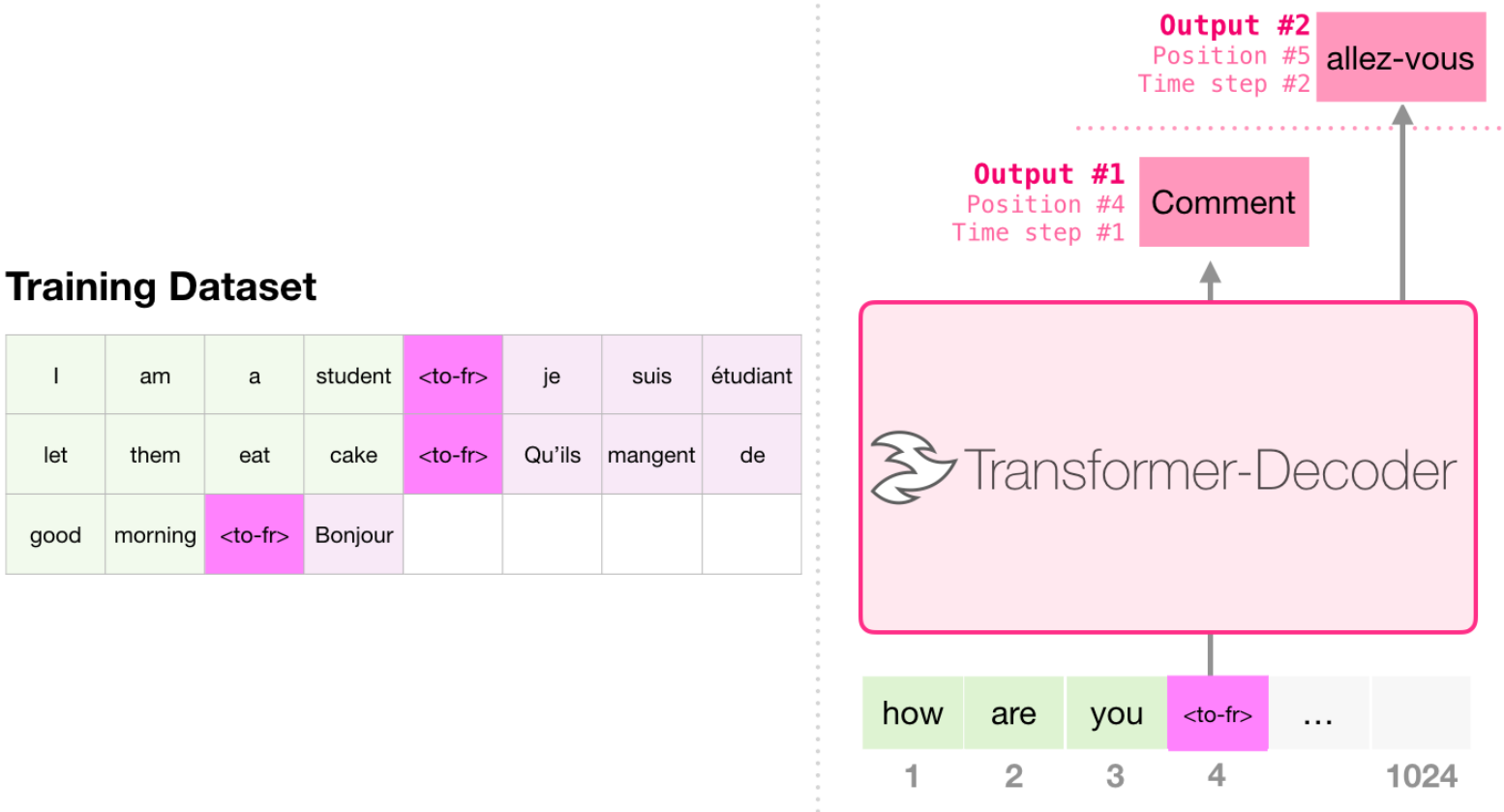
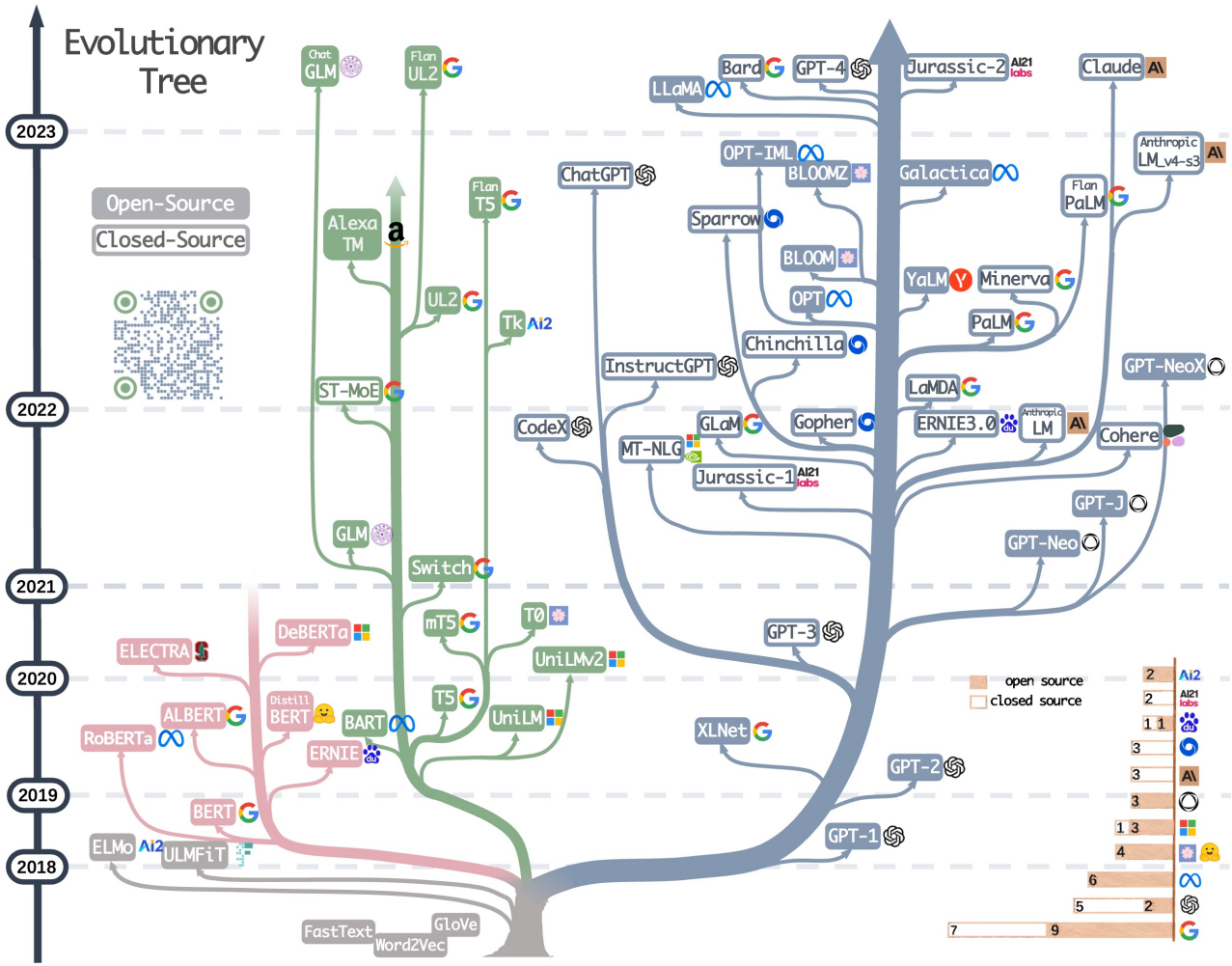


Figure: GPT: **outputs one token at a time.**[1] <to-fr> can be generalised into natural language instructions.

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

Evolutionary tree of LLMs



1) Yang, Jingfeng, et al. "Harnessing the power of llms in practice: A survey on chatgpt and beyond." arXiv preprint arXiv:2304.13712 (2023).

Scaling law

Scaling up the amount of data D, compute power C, and model parameter $N \Rightarrow$ **predictable smooth continued** increase in model performance (loss L).
Such power-law relationship is called a scaling law (read more here[3]).

$$f(x) = ax^{-k} + \epsilon \tag{1}$$

where ϵ denotes a deviation from the power-law function. k is the slope.

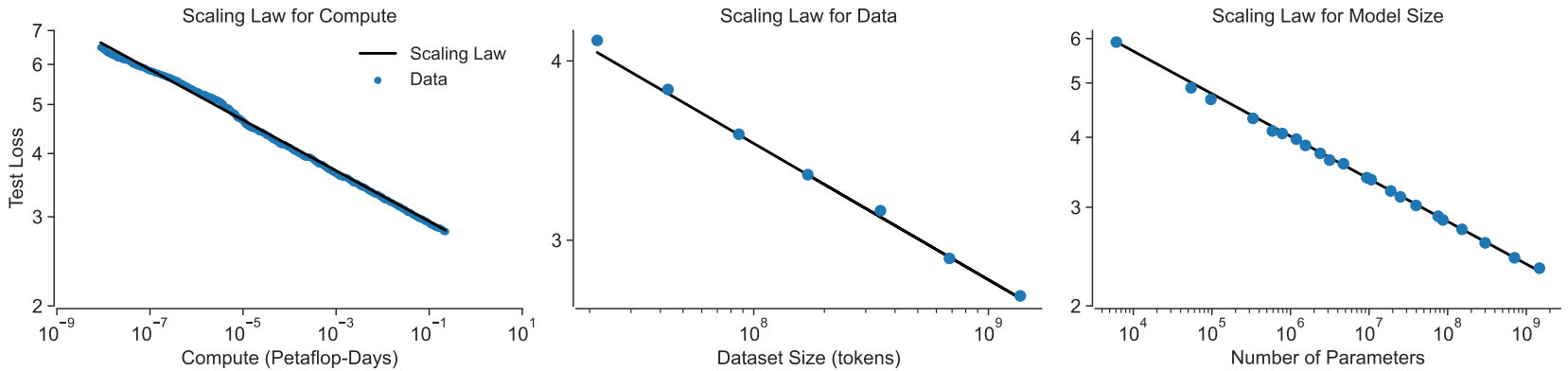


Figure: Scaling laws[1] reliably predict that model performance (y-axes) improves with increasing compute **(Left)**, training data **(Middle)**, and model size **(Right)**. In all cases a power-law (straight line, black) fits the empirically observed data (blue) exceptionally well. Figure adapted from 2. Note that x-axis and y-axis are logarithmic.

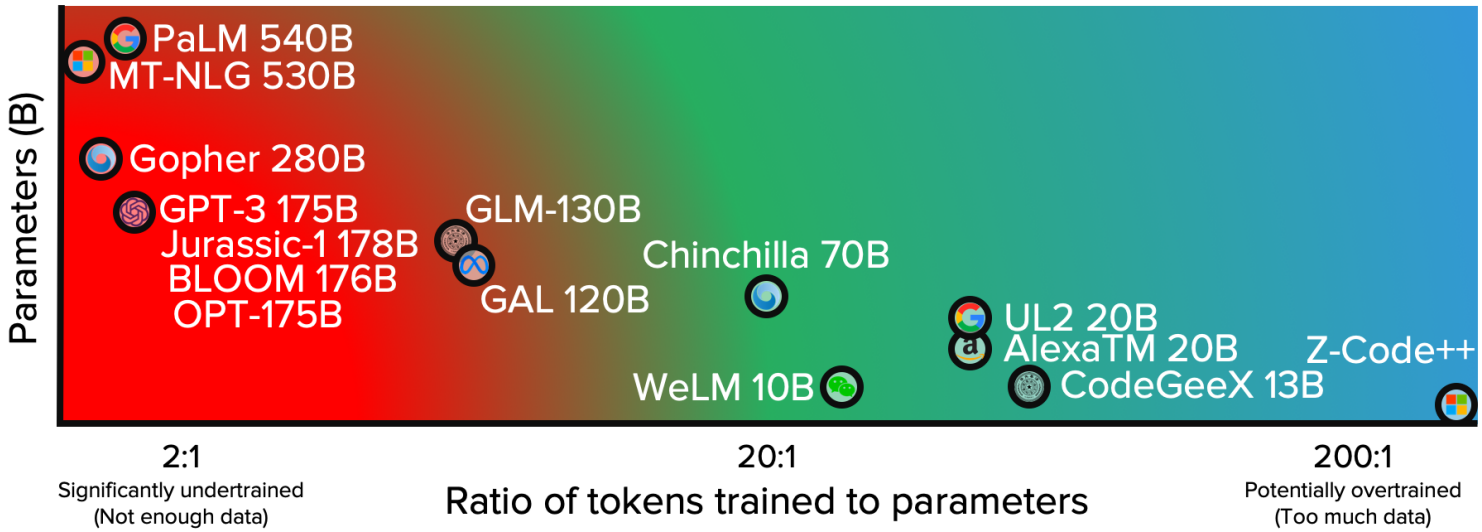
1) Ganguli, Deep, et al. "Predictability and surprise in large generative models." Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency. 2022.

2) Kaplan, Jared, et al. "Scaling laws for neural language models." arXiv preprint arXiv:2001.08361 (2020).

3) `https://en.wikipedia.org/wiki/Neural_scaling_law`

Scaling law

Chinchilla (scaling) law[1,2] suggests we need 20 text tokens per parameter, i.e., 1.4T tokens for training a 70B model.



Is scaling of $D \uparrow$, $C \uparrow$, $N \uparrow$ unlimited, as it's predictable?
By one estimate[2], the world's total stock of usable text data is between 4.6T and 17.2T tokens.

---

1) `https://lifearchitect.ai/chinchilla/`

2) `https://www.lesswrong.com/posts/6Fpvch8RR29qLEWNH/chinchilla-s-wild-implications`

2) Villalobos, Pablo, et al. "Will we run out of data? An analysis of the limits of scaling datasets in Machine Learning." arXiv preprint arXiv:2211.04325 (2022).

Emergent abilities

Abrupt jumps in certain known capabilities when increasing model size, same for unknown capabilities which are appeared to be unpredictably **emergent**[1].

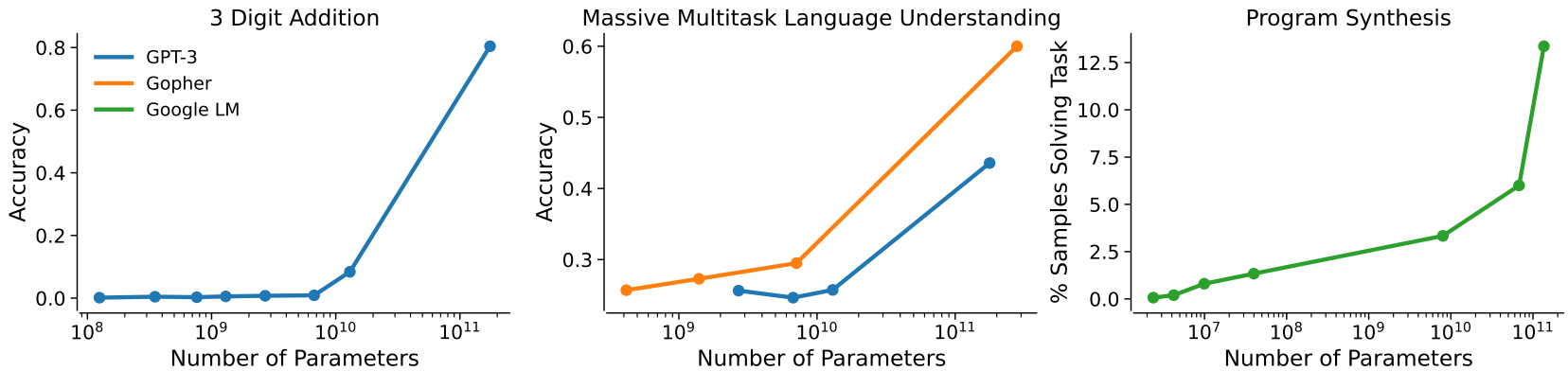


Figure: Three examples of abrupt specific capability scaling, based on three different models: GPT-3 (blue), Gopher (orange), and a Google language model (green). **(Left)** 3-Digit addition with GPT-3. **(Middle)** Language understanding with GPT-3 and Gopher. **(Right)** Program synthesis with Google language models.

Three digit addition is performed accurately less than 1% of the time on any model with less than 6B parameters, but this jumps to 8% accuracy on a 13B parameter model and 80% accuracy on a 175B parameter model.

AI safety: societally impactful potential harms (e.g., toxicity) of the model also scale.

1) Wei, Jason, et al. "Emergent abilities of large language models." arXiv preprint arXiv:2206.07682 (2022).

Chapter 3

InstructGPT[1] $\Rightarrow$ ChatGPT

- There is as much manual labor as there is intelligence.
- InstructGPT = GPT (pre-training) + Instruction-tuning (multi-task fine-tuning) + Alignment-tuning (RLHF: Reinforcement Learning from Human Feedback)

Assumption

Instructions

Alignments

1) Ouyang, Long, et al. "Training language models to follow instructions with human feedback." Advances in Neural Information Processing Systems 35 (2022): 27730-27744.

Assumption

- GPT is trained for predicting the next token on a webpage from the internet.
  $\Rightarrow$ However, what we want is "To follow the user's instructions (*instruction-tuning*), be helpful, honest, and harmless."
- Making language models bigger does not inherently make them better at following a user's intent.
- We say that the language modeling objective (CLM) is **misaligned**.

Solution: Fine-tuning to learn (to be aligned with) human feedbacks/perefences $\Rightarrow$ Alignment-tuning (RLHF: Reinforcement Learning from Human Feedback)

Human evaluation

- outputs from the 1.3B InstructGPT model are preferred to outputs from the 175B GPT-3, despite having 100x fewer parameters.

Training steps

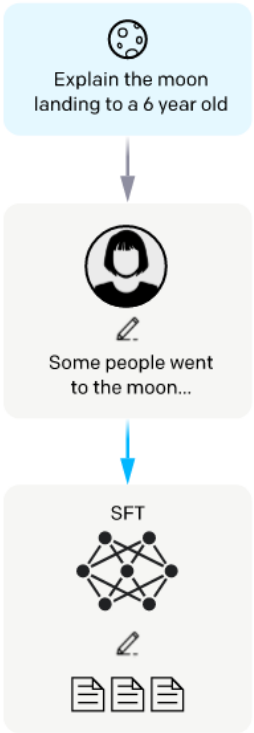
Step 1

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.



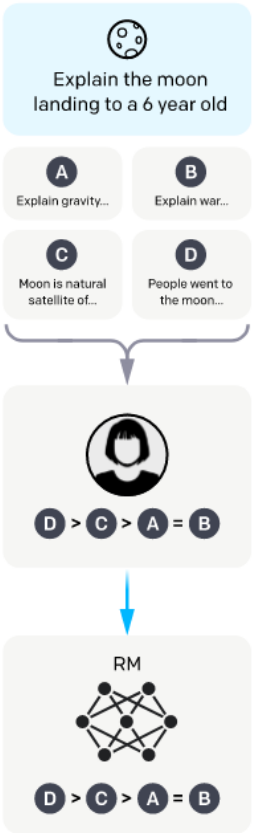
Step 2

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

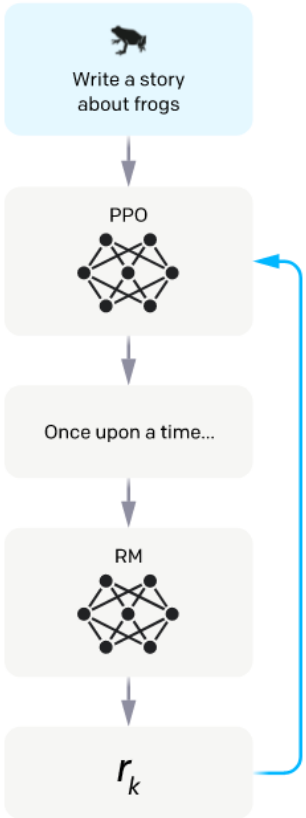
Optimize a policy against the reward model using reinforcement learning.

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.



1) https://openai.com/blog/chatgpt

Step 1: Instruction-tuning GPT-3 $\Rightarrow$ SFT model | 13k data

- **Plain**: We simply ask the labelers to come up with an arbitrary task, while ensuring the tasks had sufficient diversity.
- **Few-shot**: We ask the labelers to come up with an instruction, and multiple query/response pairs for that instruction.
- **User-based**: We had a number of use-cases stated in waitlist applications to the OpenAI API. We asked labelers to come up with prompts corresponding to these use cases.

Table: Distribution of use case categories from our API prompt dataset.

Use-case	(%)
Generation	45.6%
Open QA	12.4%
Brainstorming	11.2%
Chat	8.4%
Rewrite	6.6%
Summarization	4.2%
Classification	3.5%
Other	3.5%
Closed QA	2.6%
Extract	1.9%

Table: Illustrative prompts from our API prompt dataset. These are fictional examples inspired by real usage.

Use-case	Prompt
Brainstorming	List five ideas for how to regain enthusiasm for my career
Generation	Write a short story where a bear goes to the beach, makes friends with a seal, and then returns home.
Rewrite	This is the summary of a Broadway play: """ {summary} """ This is the outline of the commercial for that play: """

We hired a team of about 40 contractors. Inter-annotator agreement: 72.6 ± 1.5%

Step 2: training reward model (RM) | 33k data

RM $r_\theta(x, y)$ outputs a scalar reward indicating human preference, for a given pair of prompt x and response y, with parameters θ.

TRAINING:
For a given prompt x, we present labelers with anywhere between K = 4 and K = 9 responses $y_{k=1 \ldots K}$ to **rank** (*human preferences are expressed as comparisons*), e.g., $D > C > A = B$ in the figure.

The loss function for the RM is:

$$\text{loss}(\theta) = -\frac{1}{\binom{K}{2}} E_{(x, y_w, y_l) \sim D} \left[\log \left(\sigma \left(r_\theta \left(x, y_w\right) - r_\theta \left(x, y_l\right)\right)\right)\right]$$

where y_w is the preferred response out of the pair of y_w and y_l, and D is the dataset of human comparisons.

Step 3: PPO training SFT with RM $\Rightarrow$ InstructGPT | 31k data

PPO: Proximal Policy Optimization Algorithms[1]

$$\text{objective}(\phi) = E_{(x,y)\sim D_{\pi_\phi^{\mathrm{RL}}}}\left[r_\theta(x,y) - \beta \log\left(\pi_\phi^{\mathrm{RL}}(y \mid x)/\pi^{\mathrm{SFT}}(y \mid x)\right)\right] + \gamma E_{x\sim D_{pretrain}}\left[\log(\pi_\phi^{\mathrm{RL}}(x))\right]$$

where π_ϕ^{RL} is the learned RL policy, π^{SFT} is the supervised trained model, and $D_{pretrain}$ is the pretraining distribution. The KL reward coefficient, β, and the pretraining loss coefficient, γ, control the strength of the KL penalty and pretraining gradients respectively.

Alignment tax: during RL fine-tuning we mix in a small fraction of the original data used to train GPT-3, and train on this data using the normal log likelihood maximization.

Contribution

Using RLHF to align the behavior of GPT-3 to the **stated preferences** of a **specific group of people**.

1) Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

Chapter 4

Prompt engineering[1]

- Prompt engineer?
- The ways to communicate with the models.
- Better prompting $\Rightarrow$ better results.

Zero-shot | Few-shot | CoT

1) https://cameronrwolfe.substack.com/p/advanced-prompt-engineering

Zero-shot

1. **Simple prompt** (**Zero-shot**)
   `In summary: ... | To summarize in plain language, ... | The main point to take from this article is that ...`
   ⇒ Different formulations of the same prompt which might sound similar to humans can lead to generations that are quite different from each other.
2. **Adding task description**

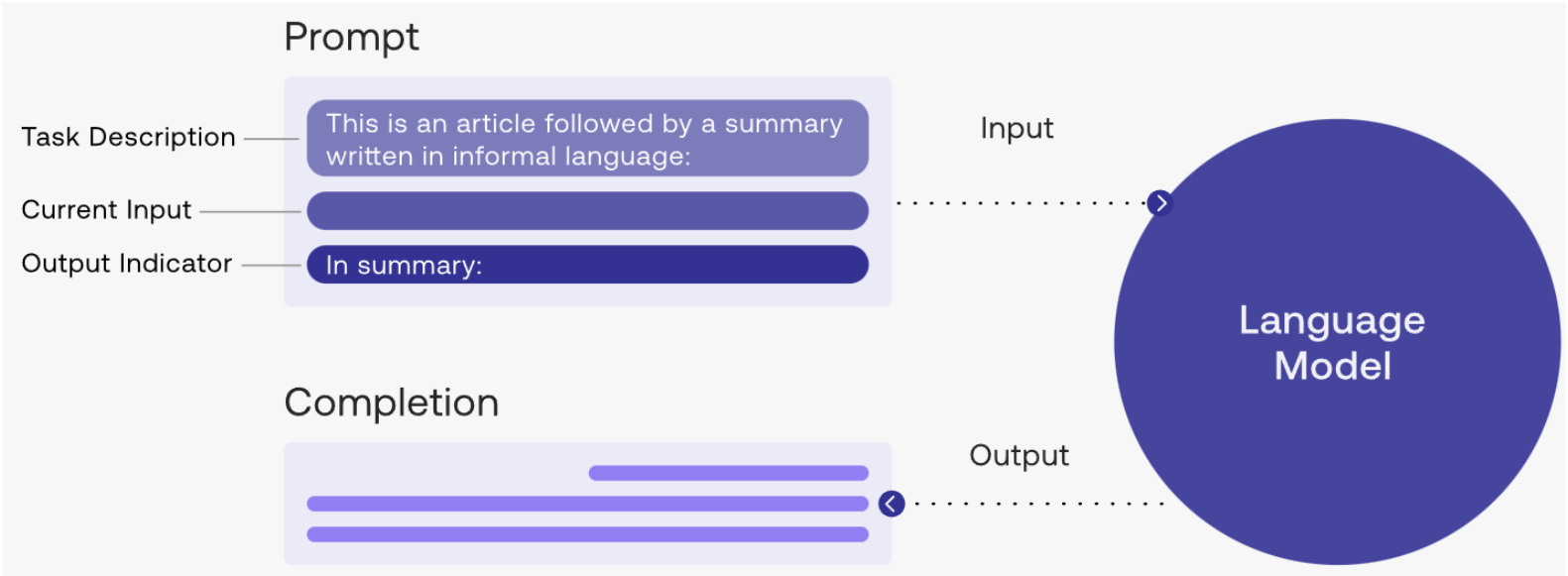


Figure: adding task description[1].

1) `https://docs.cohere.ai/docs/prompt-engineering`

Few-shot

3. **Adding examples** (**Few-shot** / in-context learning)

```
This is a movie review sentiment classifier.

Review:  "I loved this movie!" This review is positive.
Review:  "I don't know, it was ok I guess.." This review is neutral.
Review:  "What a waste of time, would not recommend this movie."
This review is negative.

Review:  "I really enjoyed this movie!" This review is ...
```

4. **Chain-of-Thought**[1] (**Few-shot-CoT**)
   A chain of thought is a series of intermediate natural language reasoning steps that lead to the final output $\Rightarrow$ improves the complex reasoning capability of LLMs.

   Note that CoT prompting is an emergent ability of model scale, it only yields performance gains when used with models of sufficient scale (100B+ parameters).

1) Wei, Jason, et al. "Chain of thought prompting elicits reasoning in large language models." arXiv preprint arXiv:2201.11903 (2022).

Few-shot Chain-of-Thought

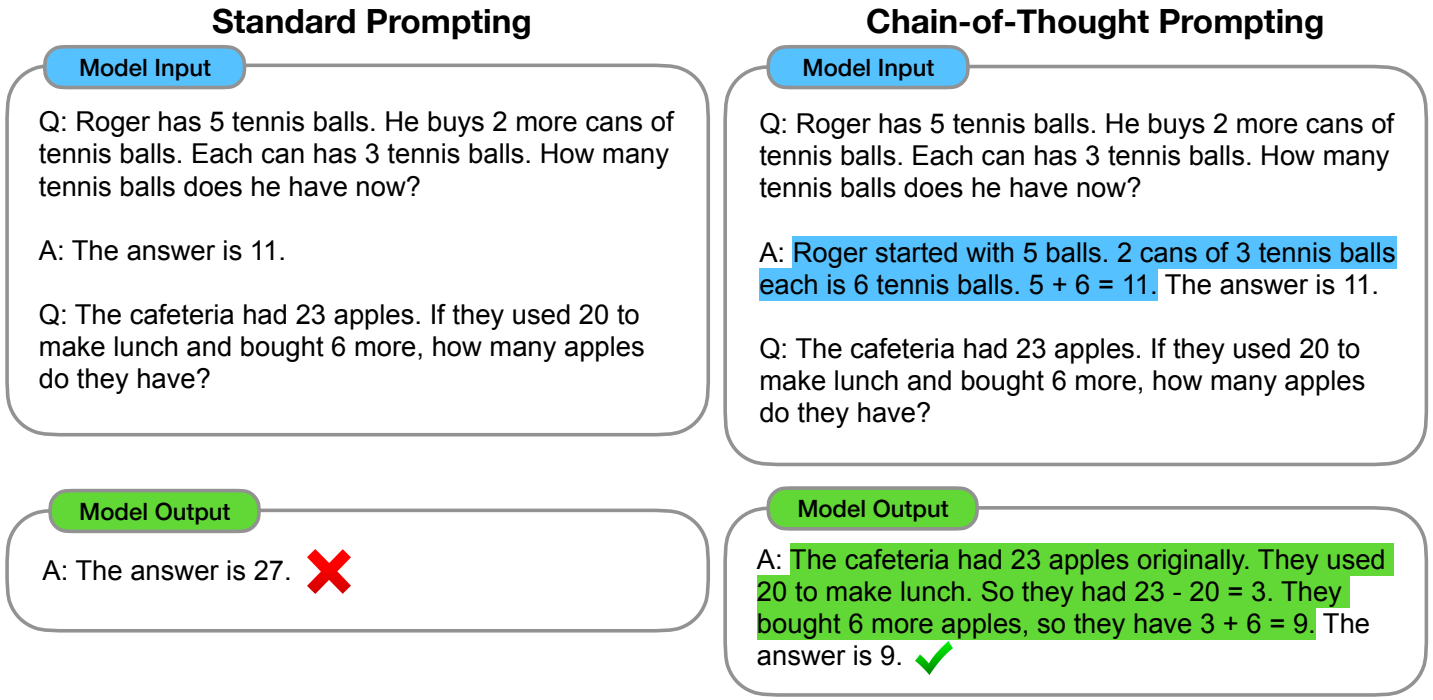


Figure: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

Self-consistency[1]: generate multiple chains of thought and taking the majority vote of these multiple outputs as the final answer.

---

1) Wang, Xuezhi, et al. "Self-consistency improves chain of thought reasoning in language models." arXiv preprint arXiv:2203.11171 (2022).

Zero-shot Chain-of-Thought

5 Let's think step by step. (Zero-shot-CoT)[1]

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The answer is 8. ✗

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. The answer is 4. ✓

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: The answer (arabic numerals) is

(Output) 8 ✗

(d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Figure: Example inputs and outputs of GPT-3 with (a) standard Few-shot, (b) Few-shot-CoT, (c) standard Zero-shot, and (d) ours (Zero-shot-CoT). Similar to Few-shot-CoT, Zero-shot-CoT facilitates multi-step reasoning (blue text) and reach correct answer where standard prompting fails. Unlike Few-shot-CoT using step-by-step reasoning examples **per task**, ours does not need any examples and just uses the same prompt "Let's think step by step" *across all tasks* (arithmetic, symbolic, commonsense, and other logical reasoning tasks).

Comparisons

Table: Comparison with baseline methods using accuracies on MultiArith and GSM8K. text-davinci-002 is used as the model if not specified. We used the same 8 examples for Few-shot and Few-shot-CoT settings. (*1) To verify the variance of changing examples, we report two results for 4-shot-cot by splitting the eight examples into two groups. (*2) We insert "Let's think step by step." at the beginning of answer part of each exemplars for Few-shot-CoT to test performance gains.

	MultiArith	GSM8K
Zero-Shot	**17.7**	**10.4**
Few-Shot (2 samples)	33.7	15.6
Few-Shot (8 samples)	33.8	15.6
Zero-Shot-CoT	**78.7**	**40.7**
Few-Shot-CoT (2 samples)	84.8	41.3
Few-Shot-CoT (4 samples : First) (*1)	89.2	-
Few-Shot-CoT (4 samples : Second) (*1)	90.5	-
Few-Shot-CoT (8 samples)	93.0	48.7
Zero-Plus-Few-Shot-CoT (8 samples) (*2)	**92.8**	**51.5**
Finetuned GPT-3 175B	-	33
Finetuned GPT-3 175B + verifier	-	55
PaLM 540B: Zero-Shot	**25.5**	**12.5**
PaLM 540B: Zero-Shot-CoT	**66.1**	**43.0**
PaLM 540B: Zero-Shot-CoT + self consistency	**89.0**	**70.1**
PaLM 540B: Few-Shot	-	17.9
PaLM 540B: Few-Shot-CoT	-	56.9
PaLM 540B: Few-Shot-CoT + self consistency	-	74.4

1) Kojima, Takeshi, et al. "Large language models are zero-shot reasoners." Advances in neural information processing systems 35 (2022): 22199-22213.

Chapter 5

Parameter-Efficient Fine-tuning

- **Fine-tuning** is a process that takes a model that has already been trained for one given task (or general purpose) and then tunes the model to make it perform a second (or specific) task.
- **Full** fine-tuning modifies all the parameters and therefore necessitates storing a full copy for each task.
- **Parameter-Efficient** fine-tuning, which freezes all or most of the pretrained parameters, and augments the model with small trainable modules (and not forget to mention one line of research considers removing parameters[1,2]).

Adapter | Prefix | LoRA

1) Radiya-Dixit, Evani, and Xin Wang. "How fine can fine-tuning be? learning efficient language models." International Conference on Artificial Intelligence and Statistics. PMLR, 2020.

2) Zhao, Mengjie, et al. "Masking as an efficient alternative to finetuning for pretrained language models." arXiv preprint arXiv:2004.12406 (2020).

Adapter-tuning

Idea: Injecting lightweight trainable modules (called the ***adapter***[1]) between frozen layers of a pre-trained model.

- within the adapter, the 1st linear layer plays a role of dimension **bottleneck** $\Rightarrow$ limit the total number of parameters of the trainable module

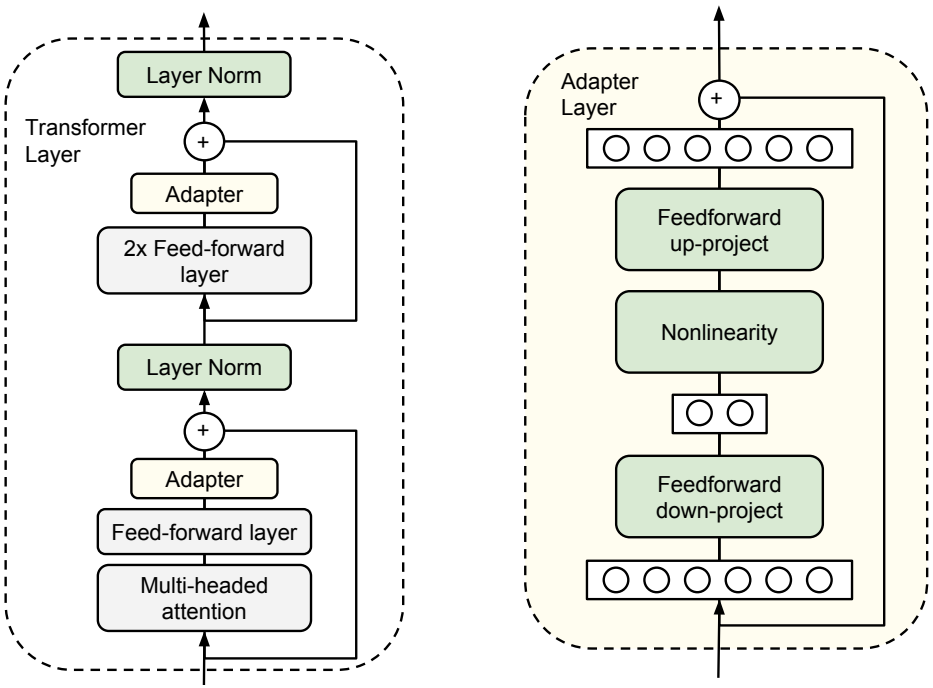


Figure: Architecture of the adapter module and its integration with the Transformer. **Left:** We add the adapter module twice to each Transformer layer: after the projection following multi-headed attention and after the two feed-forward layers. **Right:** The adapter consists of a bottleneck which contains few parameters relative to the attention and feedforward layers in the original model. The adapter also contains a skip-connection. During adapter tuning, the green layers are trained on the downstream data, this includes the adapter, the layer normalization parameters, and the final classification layer (not shown in the figure).

1) Houlsby, Neil, et al. "Parameter-efficient transfer learning for NLP." International Conference on Machine Learning. PMLR, 2019.

Adapter-tuning

+

- on GLUE benchmark[1] with BERT model, this approach attains within 0.4% of the performance of full fine-tuning, adding only 3.6% trainable parameters per task.
- another use case: ASR + mBART $\xrightarrow{\text{adapters as glue}}$ multilingual speech translation[2]

-

- additional inference latency
- posing a trade-off between efficiency and model quality.

1) `https://gluebenchmark.com/`

2) Le, Hang, et al. "Lightweight adapter tuning for multilingual speech translation." arXiv preprint arXiv:2106.01463 (2021).

Prefix-tuning

Idea: Prepending the embeddings of the input tokens with a sequence of trainable vectors (called the ***prefix***[1,2]).

- the concept originates from prompt engineering: tweaking the input phrasing to achieve better performance
  $\Rightarrow$ **hard** prompt tuning, changes the discrete input tokens (manually or autoprompt[3]), not differentiable, sub-optimal, sensitive.
- the Transformer can attend to the prefix as if it were a sequence of "virtual tokens", but unlike hard prompting, the prefix consists entirely of free parameters which do not correspond to real tokens
  $\Rightarrow$ **soft** prompt tuning, more expressive.

---

1) Li, Xiang Lisa, and Percy Liang. "Prefix-tuning: Optimizing continuous prompts for generation." arXiv preprint arXiv:2101.00190 (2021).

2) `https://lightning.ai/pages/community/article/understanding-llama-adapters/`

3) Shin, Taylor, et al. "Autoprompt: Eliciting knowledge from language models with automatically generated prompts." arXiv preprint arXiv:2010.15980 (2020).

Prefix-tuning

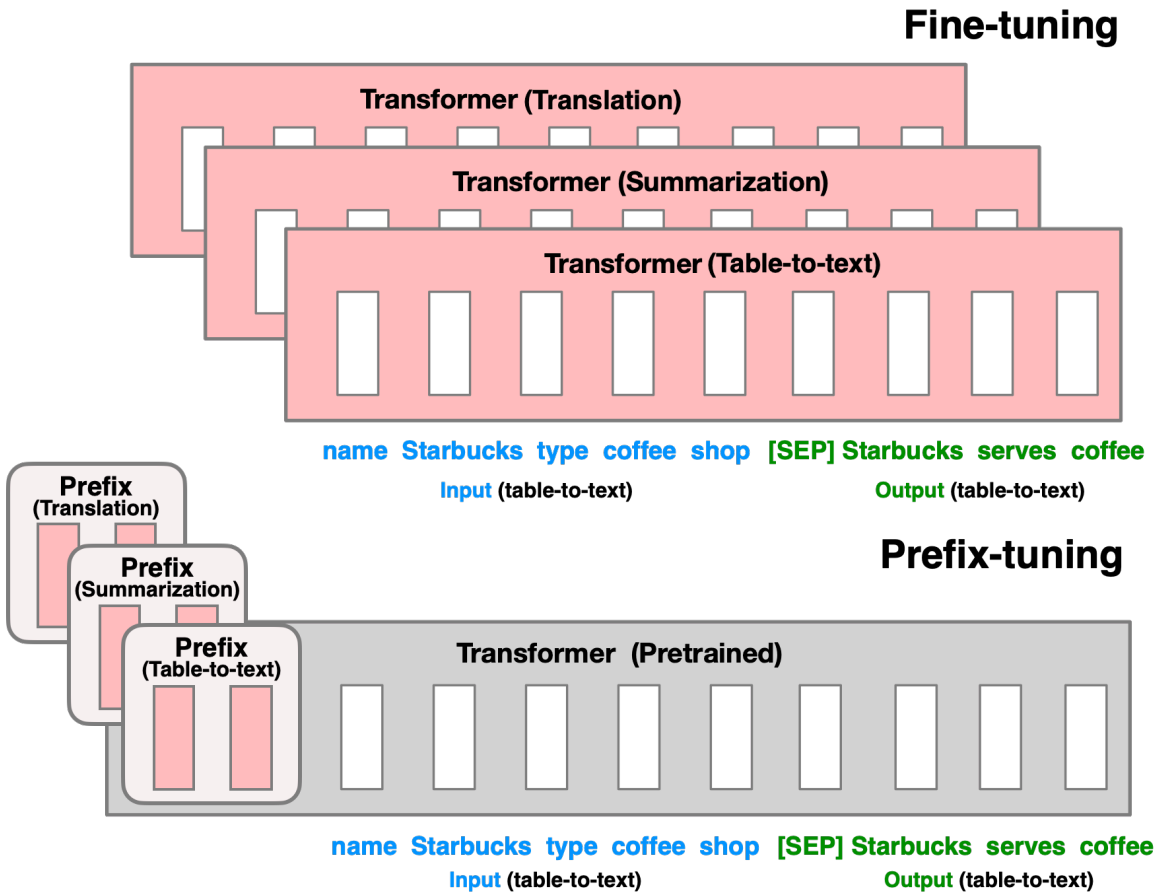


Figure: Full fine-tuning (top) updates all Transformer parameters (the red Transformer box) and requires storing a full model copy for each task. We propose prefix-tuning (bottom), which freezes the Transformer parameters and only optimizes the prefix (the red prefix blocks). Consequently, we only need to store the prefix for each task, making prefix-tuning modular and space-efficient. Note that each vertical block denote transformer activations at one time step.

Prefix-tuning

	E2E					WebNLG									DART					
	BLEU	NIST	MET	R-L	CIDEr	BLEU			MET			TER ↓			BLEU	MET	TER ↓	Mover	BERT	BLEURT
						S	U	A	S	U	A	S	U	A						
									GPT-2_{MEDIUM}											
FINE-TUNE	68.2	8.62	**46.2**	71.0	2.47	**64.2**	27.7	46.5	**0.45**	0.30	0.38	**0.33**	0.76	0.53	46.2	**0.39**	**0.46**	**0.50**	**0.94**	**0.39**
FT-TOP2	68.1	8.59	46.0	70.8	2.41	53.6	18.9	36.0	0.38	0.23	0.31	0.49	0.99	0.72	41.0	0.34	0.56	0.43	0.93	0.21
ADAPTER(3%)	68.9	8.71	46.1	71.3	2.47	60.4	**48.3**	54.9	0.43	**0.38**	**0.41**	0.35	**0.45**	**0.39**	45.2	0.38	**0.46**	**0.50**	**0.94**	**0.39**
ADAPTER(0.1%)	66.3	8.41	45.0	69.8	2.40	54.5	45.1	50.2	0.39	0.36	0.38	0.40	0.46	0.43	42.4	0.36	0.48	0.47	**0.94**	0.33
PREFIX(0.1%)	**69.7**	**8.81**	46.1	**71.4**	**2.49**	62.9	45.6	**55.1**	0.44	**0.38**	**0.41**	0.35	0.49	0.41	**46.4**	0.38	**0.46**	**0.50**	**0.94**	**0.39**
									GPT-2_{LARGE}											
FINE-TUNE	68.5	8.78	46.0	69.9	2.45	**65.3**	43.1	55.5	**0.46**	0.38	**0.42**	**0.33**	0.53	0.42	**47.0**	**0.39**	0.46	**0.51**	**0.94**	**0.40**
Prefix	**70.3**	**8.85**	**46.2**	**71.7**	**2.47**	63.4	**47.7**	**56.3**	0.45	**0.39**	**0.42**	0.34	**0.48**	**0.40**	46.7	**0.39**	**0.45**	**0.51**	**0.94**	**0.40**
SOTA	68.6	8.70	45.3	70.8	2.37	63.9	52.8	57.1	0.46	0.41	0.44	-	-	-	-	-	-	-	-	-

Table: Metrics (higher is better, except for TER) for table-to-text generation on E2E (left), WebNLG (middle) and DART (right). With only 0.1% parameters, Prefix-tuning outperforms other baselines and achieves a comparable performance with fine-tuning. The best score is boldfaced for both GPT-2_{MEDIUM} and GPT-2_{LARGE}.

+

- prefix-tuning is significantly better than adapter (0.1%), attaining 4.1 BLEU improvement per dataset on average.
- even when comparing with fine-tuning (100%) and adapter-tuning (3.0%), which update significantly more parameters than prefix-tuning, prefix-tuning still achieves results comparable or better than those two systems.

Prefix-tuning

The performance increases as the prefix length increases up to a threshold (200 for summarization, 10 for table-to-text) and then a slight performance drop occurs.

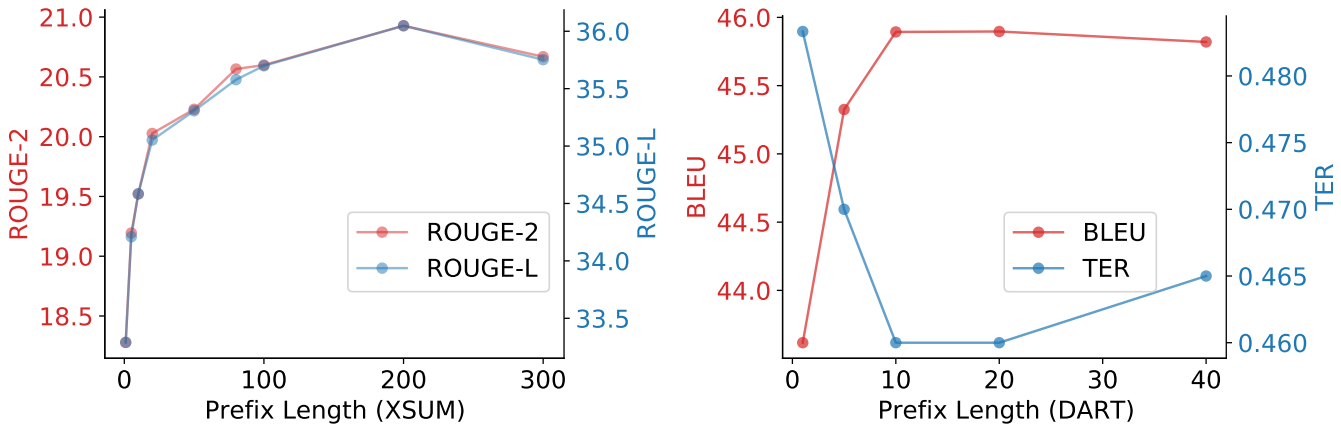


Figure: Prefix length vs. performance on summarization (left) and table-to-text (right). Each plot reports two metrics (on two vertical axes).

Initializing the prefix with task-related words significantly improves generation. "summarize" $>$ "elephant" $>$ random

-

- it's difficult to optimize, its performance changes non-monotonically in trainable parameters.[3]
- it takes context space.

LoRA

Low-Rank Adaptation[1] freezes the pre-trained model weights W and injects trainable rank decomposition matrices A, B into each layer of the Transformer $\Rightarrow W + BA$.

- the learned over-parametrized models in fact reside on a **low intrinsic dimension**[2,3]. We hypothesize that the change in weights during model adaptation also has a low "intrinsic rank".

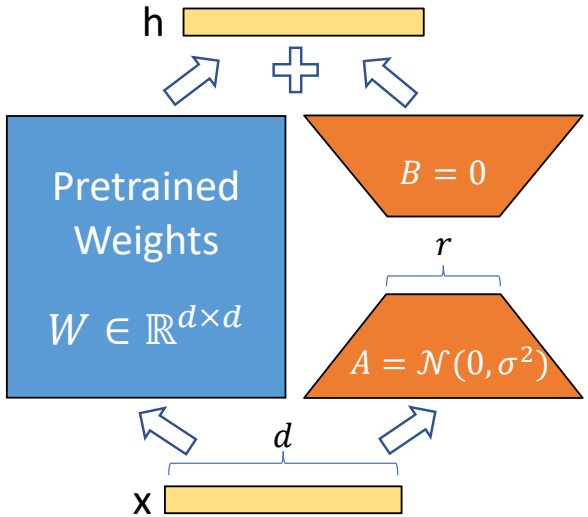


Figure: The reparametrization. We only train A and B. ***r* can be one or two.**

1) Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." arXiv preprint arXiv:2106.09685 (2021).

2) Li, Chunyuan, et al. "Measuring the intrinsic dimension of objective landscapes." arXiv preprint arXiv:1804.08838 (2018).

3) Aghajanyan, Armen, Luke Zettlemoyer, and Sonal Gupta. "Intrinsic dimensionality explains the effectiveness of language model fine-tuning." arXiv preprint arXiv:2012.13255 (2020).

LoRA

A pre-trained GPT $P_\Phi(y|x)$ parametrized by Φ. A fine-tuning dataset of context-target pairs: $\mathcal{Z} = \{(x_i, y_i)\}_{i=1,..,N}$, where both x_i and y_i are sequences of tokens.
During full fine-tuning, the model is initialized to pre-trained weights Φ_0 and updated to $\Phi_0 + \Delta\Phi$ by repeatedly following the gradient to maximize the conditional language modeling objective:

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_\Phi(y_t|x, y_{<t})\right)$$

One of the main drawbacks for full fine-tuning is that for **each** downstream task, we learn a **different** set of parameters $\Delta\Phi$ whose dimension $|\Delta\Phi|$ equals $|\Phi_0|$.
We adopt a more parameter-efficient approach, where the task-specific parameter increment $\Delta\Phi = \Delta\Phi(\Theta)$ is further encoded by a much smaller-sized set of parameters Θ with $|\Theta| \ll |\Phi_0|$. The task of finding $\Delta\Phi$ thus becomes optimizing over Θ:

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(p_{\Phi_0+\Delta\Phi(\Theta)}(y_t|x, y_{<t})\right)$$

We propose to use a low-rank representation to encode $\Delta\Phi$ that is both compute- and memory-efficient. When the pre-trained model is GPT-3 175B, the number of trainable parameters $|\Theta|$ can be as small as 0.01% of $|\Phi_0|$.

LoRA

For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, we constrain its update by representing the latter with a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}$, and the rank $r \ll \min(d, k)$.

During training, W_0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. Note both W_0 and $\Delta W = BA$ are multiplied with the same input, and their respective output vectors are summed coordinate-wise. For $h = W_0 x$, our modified forward pass yields:

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

We use a random Gaussian initialization for A and zero for B, so $\Delta W = BA$ is zero at the beginning of training.

LoRA

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

Table: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around $\pm 0.5\%$, MNLI-m around $\pm 0.1\%$, and SAMSum around $\pm 0.2/\pm 0.2/\pm 0.1$ for the three metrics.

A more recent study[1] also confirms its effectiveness, with hyper-parameter suggestion.

1) `https://www.anyscale.com/blog/fine-tuning-llms-lora-or-full-parameter-an-in-depth-analysis-with-llama-`

LoRA

+

- We can freeze the base model and efficiently switch different tasks by replacing the matrices A and B, reducing the storage requirement and task-switching overhead significantly.
- LoRA makes training more efficient and lowers the hardware barrier to entry by up to 3 times when using adaptive optimizers since we do not need to calculate the gradients or maintain the optimizer states for most parameters. Instead, we only optimize the injected, much smaller low-rank matrices.
- Our simple linear design allows us to merge the trainable matrices with the frozen weights when deployed, *introducing no inference latency* compared to a fully fine-tuned model, by construction.
- LoRA is orthogonal to many prior methods and can be combined with many of them, such as prefix-tuning.

Storage: A full 7B LLaMA checkpoint requires 23 GB of storage capacity, while the LoRA weights can be as small as 8 MB if we choose a rank of r=8.

VRAM: T0-3B model, full fine-tuning requires 47.14 GB, LoRA needs 14.4 GB.

Chapter 6

Quantization

- for example, just to do inference on BLOOM-176B, you would need to have 8x 80GB A100 GPUs ($15k each). To fine-tune BLOOM-176B, you'd need 72 of these GPUs![1]
- we need to find ways to reduce these requirements while preserving the model's performance.

bfloat16 | 8-bit | 4-bit

1) https://huggingface.co/blog/hf-bitsandbytes-integration

Bfloat16

The number of digits allowed to use in the **mantissa** governs the **precision** of the value, the **exponent** governs the **range**, e.g., 6.02×10^{23} v.s. $6.022140857 \times 10^{23}$.

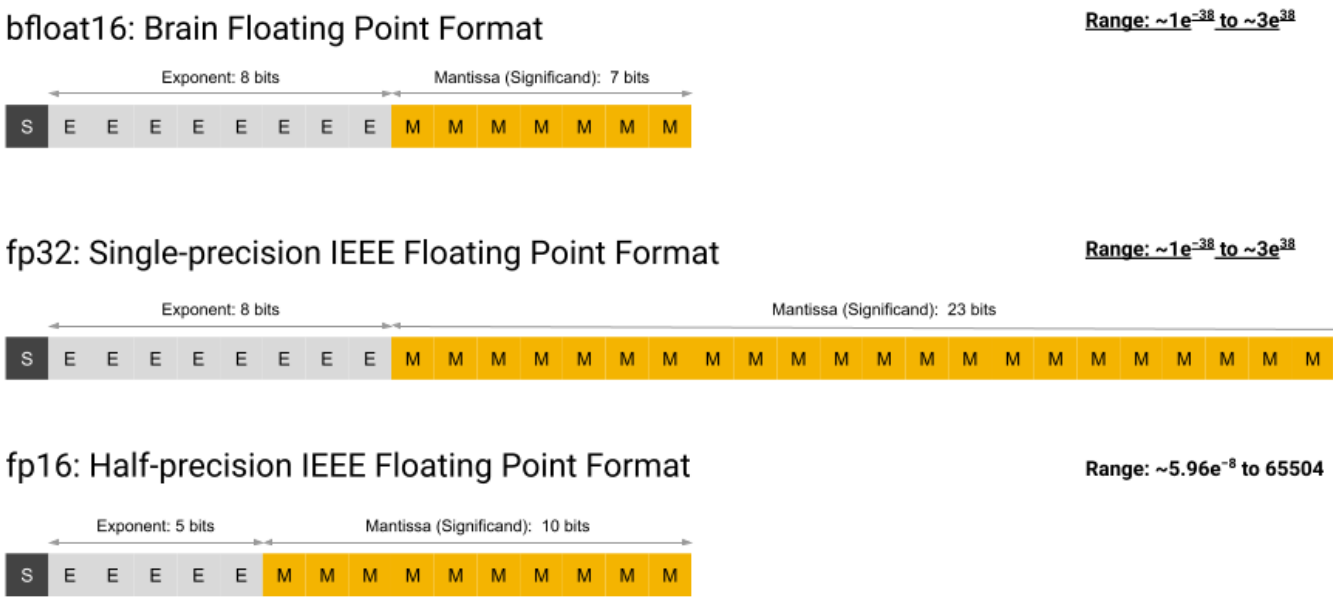


Figure: floating point formats: `bfloat16` (used by BLOOM-176B), `float16` (used by BLOOM-7.1B) and `float32`.[1]

Replacing `float32` with `bfloat16` can shorten the time, uses less memory while preserving the accuracy.[2]
(models are more sensitive to changes in exponent rather than mantissa)

1) `https://cloud.google.com/tpu/docs/bfloat16`
2) `https://www.cerebras.net/blog/to-bfloat-or-not-to-bfloat-that-is-the-question/`

Introduction

Quantization example: A non-standard 2-bit data type
Map: Index: 0, 1, 2, 3 → -1.0, 0.3, 0.5, 1.0
Input tensor: [10, -3, 5, 4]

1. Normalize with **absmax**: [10, -3, 5, 4] → [1.0, -0.3, 0.5, 0.4]
2. Find closest value: [1.0, -0.3, 0.5, 0.4] → [1.0, 0.3, 0.5, 0.5]
3. Find the associated index: [1.0, 0.3, 0.5, 0.5] → [3, 1, 2, 2] -> store
4. Dequantization: load → [3, 1, 2, 2] → lookup → [1.0, 0.3, 0.5, 0.5] → denormalize → [10, 3, 5, 5]

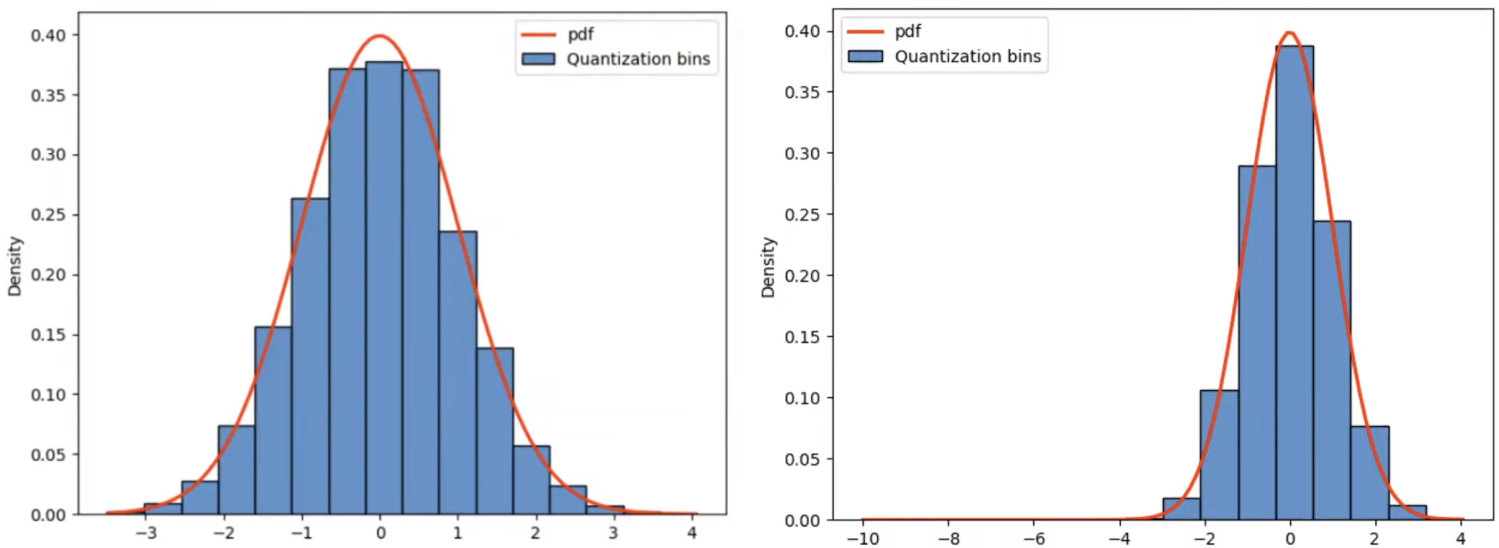


Figure: linear 4-bit quantization[1], outlier issue (at -10)

1) https://www.youtube.com/watch?v=jy0qtw4ry2w

8-bit optimizer

Stateful optimizers maintain gradient statistics over time to accelerate optimization $\Rightarrow$ these optimizer states take 33-75% of the total memory footprint during training!

$$\text{Adam}(\mathbf{g}_t, \mathbf{w}_{t-1}, \mathbf{m}_{t-1}, \mathbf{r}_{t-1}) = \begin{cases} \mathbf{r}_0 = \mathbf{m}_0 = \mathbf{0} & \text{Initialization} \\ \mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t & \text{State 1 update} \\ \mathbf{r}_t = \beta_2 \mathbf{r}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 & \text{State 2 update} \\ \mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \cdot \frac{\mathbf{m}_t}{\sqrt{\mathbf{r}_t}+\epsilon} & \text{Weight update,} \end{cases}$$

where $\mathbf{g}_t = \frac{\partial \mathbf{L}}{\partial \mathbf{w}}$ is the gradient. β_1 and β_2 are smoothing constants, ϵ is a small constant, and α is the learning rate.
For 32-bit states, Adam consume 8 bytes per parameter. That is 8 GB for a 1B parameter model. 8-bit quantization reduces the cost to 2 GB.

The work[1] develops a fast, high-precision non-linear quantization method – block-wise dynamic quantization – that enables stable 8-bit optimizers (e.g., Adam, AdamW, and Momentum) which maintain 32-bit performance at a fraction of the memory footprint and without any changes to the original hyperparameters.

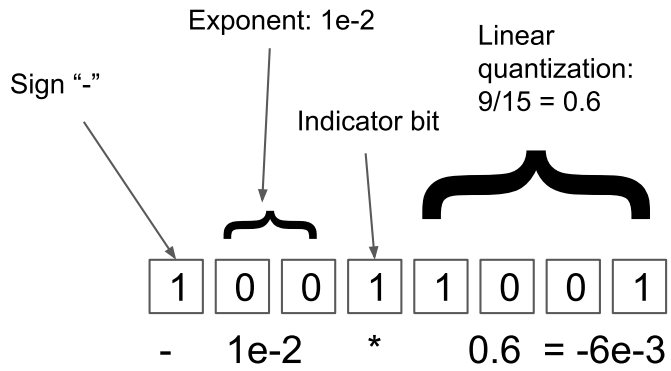


Figure: Dynamic tree quantization.

1) Dettmers, Tim, et al. "8-bit optimizers via block-wise quantization." arXiv preprint arXiv:2110.02861 (2021).

8-bit optimizer (training) and LLM.int8()[1] (inference)

`https://github.com/TimDettmers/bitsandbytes`

Model	Inference memory	Fine-tuning memory
T5-11B	22 GB	176 GB
OPT-66B	132 GB	1,056 GB
BLOOM 176B	352 GB	2,800 GB

LLM.int8() 8-bit optimizers

Model	Inference memory	Fine-tuning memory
T5-11B	11 GB	66 GB
OPT-66B	66 GB	396 GB
BLOOM 176B	176 GB	1,056 GB

Figure: VRAM reduction of 8-bit quantization.[2]

1) Dettmers, Tim, et al. "Llm.int8(): 8-bit matrix multiplication for transformers at scale." arXiv preprint arXiv:2208.07339 (2022).

2) `https://www.youtube.com/watch?v=jy0qtw4ry2w`

4-bit inference and 4-bit QLoRA

QLoRA[1,2] has one storage data type (usually 4-bit NormalFloat: `nf4`) for the base model weights and a computation data type (16-bit BrainFloat: `bfloat16`) used to perform computations.

```python
from transformers import BitsAndBytesConfig

nf4_config = BitsAndBytesConfig(
   load_in_4bit=True,
   bnb_4bit_quant_type="nf4",
   bnb_4bit_use_double_quant=True,
   bnb_4bit_compute_dtype=torch.bfloat16
)

model_nf4 = AutoModelForCausalLM.from_pretrained(
    model_id, quantization_config=nf4_config
)
```

The weights are decompressed only when they are needed for the computation, therefore the memory usage stays low during training and inference.

---

1) Dettmers, Tim, et al. "Qlora: Efficient finetuning of quantized llms." arXiv preprint arXiv:2305.14314 (2023).

2) `https://huggingface.co/blog/4bit-transformers-bitsandbytes`

Chapter 7

Transformer upgrade

- Increase input context size.[1]
- Reduce computational cost.

RoPE

AliBi

Multi-Query

1) https://www.anthropic.com/index/100k-context-windows

RoPE

- Rotary Position Embedding (RoPE)[1] is a realization of relative position encoding by means of absolute position encoding.
- Objective: $\boldsymbol{q}_m^T \boldsymbol{k}_n$ be a function g of only word embeddings $\boldsymbol{x}_m$, $\boldsymbol{x}_n$, and their relative position $m - n$.

$$\boldsymbol{q}_m^T \boldsymbol{k}_n = \langle f_q(\boldsymbol{x}_m, m), f_k(\boldsymbol{x}_n, n) \rangle = g(\boldsymbol{x}_m, \boldsymbol{x}_n, m - n)$$

  where f_q and f_k are positional embedding functions.
- 2D case:

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W_{\{q,k\}}^{(11)} & W_{\{q,k\}}^{(12)} \\ W_{\{q,k\}}^{(21)} & W_{\{q,k\}}^{(22)} \end{pmatrix} \begin{pmatrix} x_m^{(1)} \\ x_m^{(2)} \end{pmatrix}$$

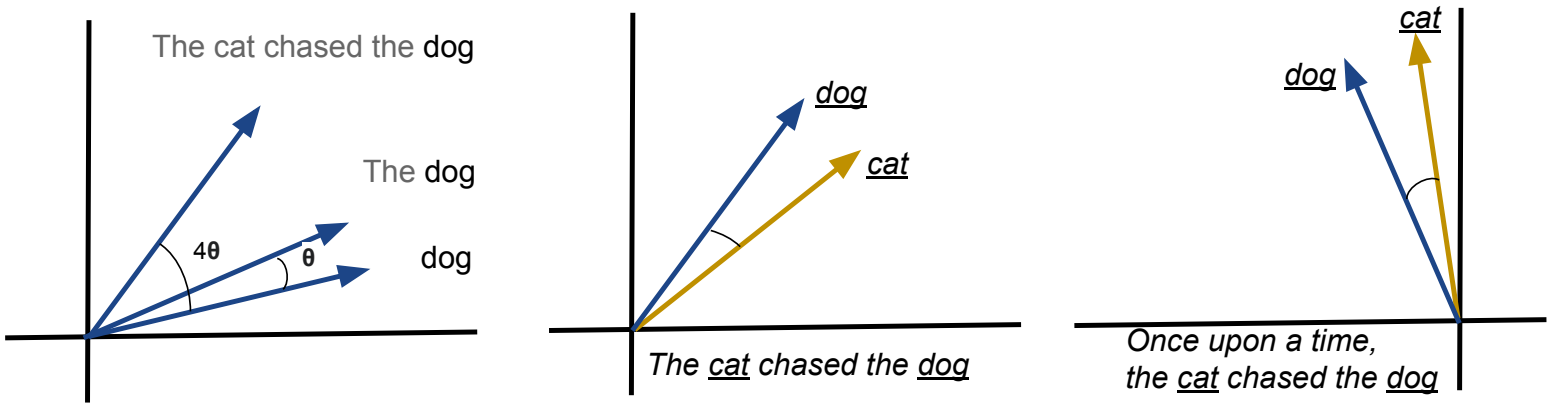
  where m is the absolute position, and θ is a predefined angle.

---

1) Su, Jianlin, et al. "Roformer: Enhanced transformer with rotary position embedding." arXiv preprint arXiv:2104.09864 (2021).

2) Chen, Shouyuan, et al. "Extending context window of large language models via positional interpolation." arXiv preprint arXiv:2306.15595 (2023).

RoPE



General form: we divide the d-dimension space into $d/2$ sub-spaces and combine them in the merit of the linearity of the inner product, turning $f_{\{q,k\}}$ into $f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m}\boldsymbol{W}_{\{q,k\}}\boldsymbol{x}_m$ where the rotary matrix:

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

with pre-defined parameters $\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}$.

ALiBi[1]

The training speed of transformer LMs gets slower as the input subsequence length L increases.

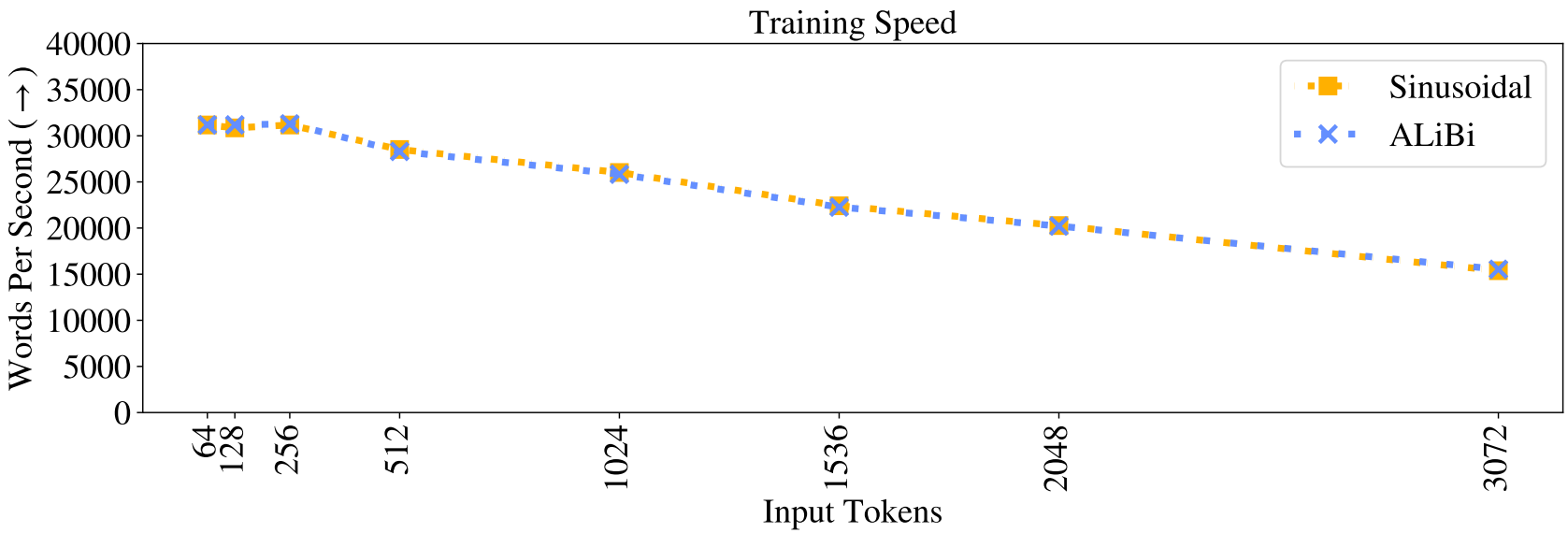


Figure: Training speed of our model and the sinusoidal baseline trained on different amounts of input subsequence tokens L.

Extrapolation: a model's ability to continue performing well as the number of input tokens during **inference** increases beyond the number of tokens on which the model was **trained**. $\Rightarrow$ determined by positional encoding.

1) Press, Ofir, Noah A. Smith, and Mike Lewis. "Train short, test long: Attention with linear biases enables input length extrapolation." arXiv preprint arXiv:2108.12409 (2021).

AliBi

Attention with Linear Biases (ALiBi): it biases query-key attention scores with a penalty that is proportional to their distance.

$$\text{softmax}(\mathbf{q}_i \mathbf{K}^\top + m \cdot [-(i-1), ..., -2, -1, 0]), \tag{2}$$

For our models with 8 heads, the slopes that we used are the geometric sequence: $\frac{1}{2^1}, \frac{1}{2^2}, ..., \frac{1}{2^8}$. The different heads increase their penalties at different rates, depending on the slope magnitude.

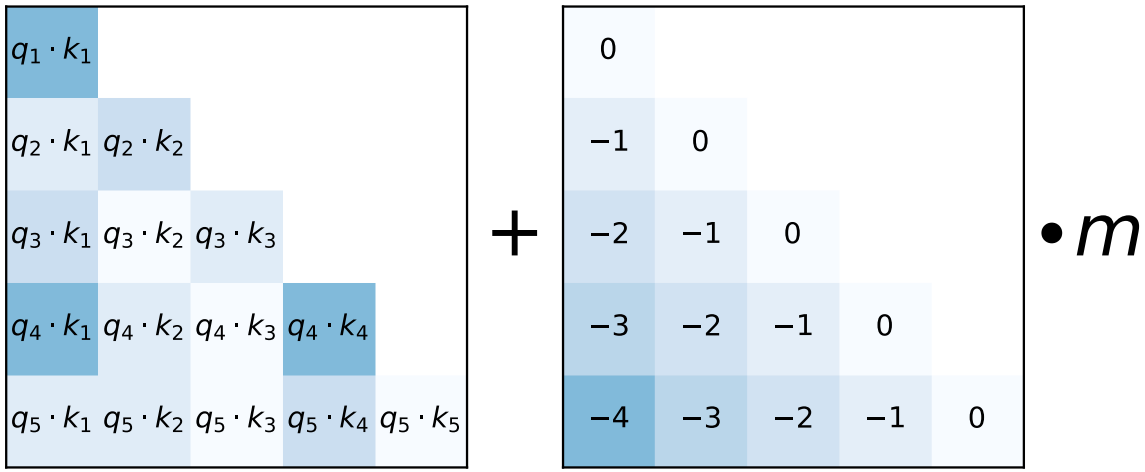


Figure: When computing attention scores for each head, our linearly biased attention method, ALiBi, adds a constant bias (right) to each attention score ($\mathbf{q}_i \cdot \mathbf{k}_j$, left). As in the unmodified attention sublayer, the softmax function is then applied to these scores, and the rest of the computation is unmodified. **m is a head-specific scalar** that is set and not learned throughout training. We show that our method for setting m values generalizes to multiple text domains, models and training compute budgets. When using ALiBi, we do **not** add positional embeddings at the bottom of the network.

AliBi

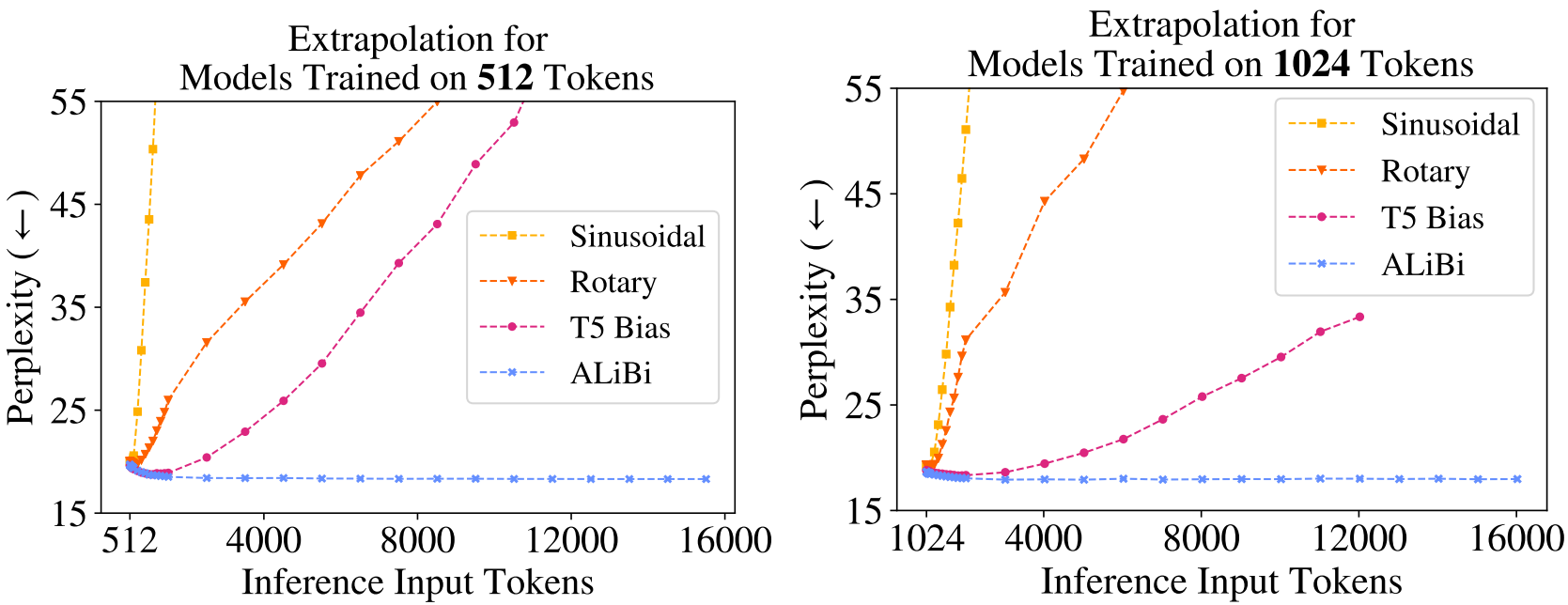


Figure: Extrapolation: as the (validation-set's) input sequence gets longer (x-axis), current position methods (sinusoidal, rotary, and T5) show degraded perplexity (y-axis, lower is better), but our method does not. Models were trained on WikiText-103 with sequences of $L = 512$ (left) or $L = 1{,}024$ (right) tokens. T5 ran out of memory on our 32GB GPU.

Multi-Query Attention

Multi-Head Attention (MHA)[1] scheme has one query, key, and value per head; **Multi-Query Attention** (MQA) instead shares one key and value across all heads.

Either training with MQA from the beginning, or convert trained MHA models to MQA architecture:

1. key/value heads are mean pooled into a single one
2. uptraining: train the converted checkpoint on a small portion of original data

Grouped-Query Attention (GQA)[2]

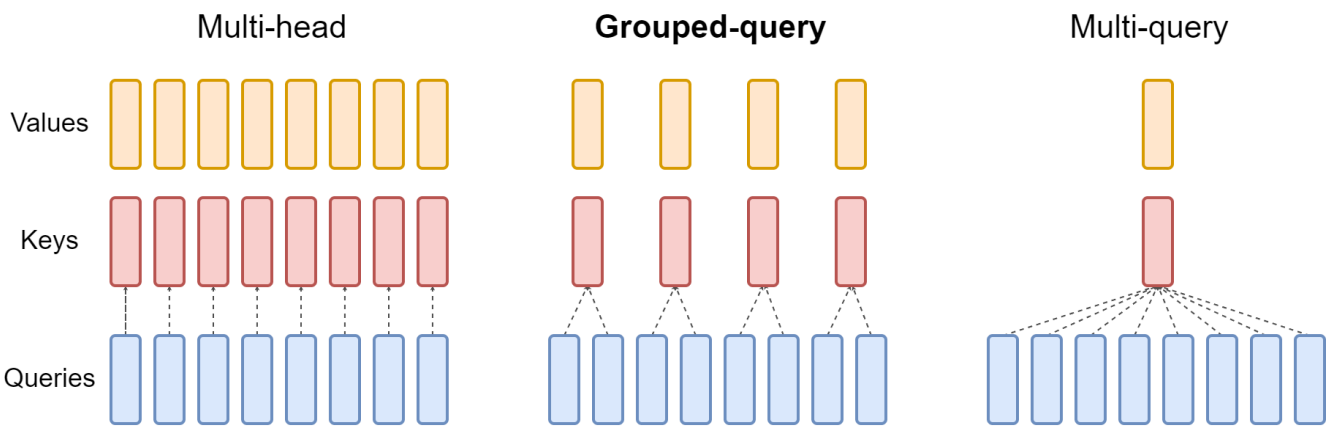


Figure: MHA has H query, key, and value heads. MQA shares single key and value heads across all query heads. GQA uses single key and value heads for each **group of query heads**, interpolating between MHA and MQA.

1) Shazeer, Noam. "Fast transformer decoding: One write-head is all you need." arXiv:1911.02150 (2019).

2) Ainslie, Joshua, et al. "GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints." arXiv preprint arXiv:2305.13245 (2023).

Chapter 8

Data & LLMs

- Data is the new oil.
- New day, new model.

Data

Augmentation

LLMs

Open pre-training data

Collection, cleaning, filtering, de-duplication, de-identifiacation.

- BigScience ROOTS[1]
  - a composite collection of 498 Hugging Face datasets
  - 46 natural languages and 13 programming languages
- RedPajama[2]
  - a reproduction of the LLaMA training dataset of over 1.2T tokens
  - V2 version[3]: 30T tokens for English, French, Spanish, German, and Italian
- Falcon RefinedWeb[4]
  - a high-quality 5T tokens web-only English pretraining dataset
- OSCAR[5]
  - 166 different languages available
- The Pile[6]

---

1) Laurençon, Hugo, et al. "The bigscience roots corpus: A 1.6 tb composite multilingual dataset." Advances in Neural Information Processing Systems 35 (2022): 31809-31826.

2) `https://www.together.xyz/blog/redpajama`

3) `https://together.ai/blog/redpajama-data-v2`

4) Penedo, Guilherme, et al. "The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only." arXiv preprint arXiv:2306.01116 (2023).

5) `https://huggingface.co/oscar-corpus`

6) Gao, Leo, et al. "The pile: An 800gb dataset of diverse text for language modeling." arXiv preprint arXiv:2101.00027 (2020).

Open instruction-tuning data

- BigScience P3 (Public Pool of Prompts)[1,2,3]
  - xP3: Mixture of 13 training tasks in 46 languages with English prompts
    $\Rightarrow$ BLOOMZ
  - xP3mt: ... with prompts in 20 languages (machine-translated from English)
    $\Rightarrow$ BLOOMZ-MT
- LAION OIG[4]
  - 43M instructions, including P3
- Databricks Dolly[5]
  - 15k instructions, authored by more than 5,000 Databricks employees

---

1) Muennighoff, Niklas, et al. "Crosslingual generalization through multitask finetuning." arXiv preprint arXiv:2211.01786 (2022).

2) Bach, Stephen H., et al. "Promptsource: An integrated development environment and repository for natural language prompts." arXiv preprint arXiv:2202.01279 (2022).

3) Sanh, Victor, et al. "Multitask prompted training enables zero-shot task generalization." arXiv preprint arXiv:2110.08207 (2021).

4) `https://laion.ai/blog/oig-dataset/`

5 `https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm`

Open alignment-tuning data

- OpenAssistant Conversations[1]
  - a human-generated, human-annotated assistant-style conversation corpus consisting of 161,443 messages distributed across 66,497 conversation trees.
  - 35 different languages, annotated with 461,292 quality ratings.
  - a product of a worldwide crowd-sourcing effort involving over 13,500 volunteers.

1) Köpf, Andreas, et al. "OpenAssistant Conversations–Democratizing Large Language Model Alignment." arXiv preprint arXiv:2304.07327 (2023).

Data augmentation

Idea: Using existing LLM to create synthetic data for training future LLM (e.g., Self-instruct[1] (alpaca model[3]), self-improve[2]) $\Rightarrow$ its repetition leads to degeneration[4].

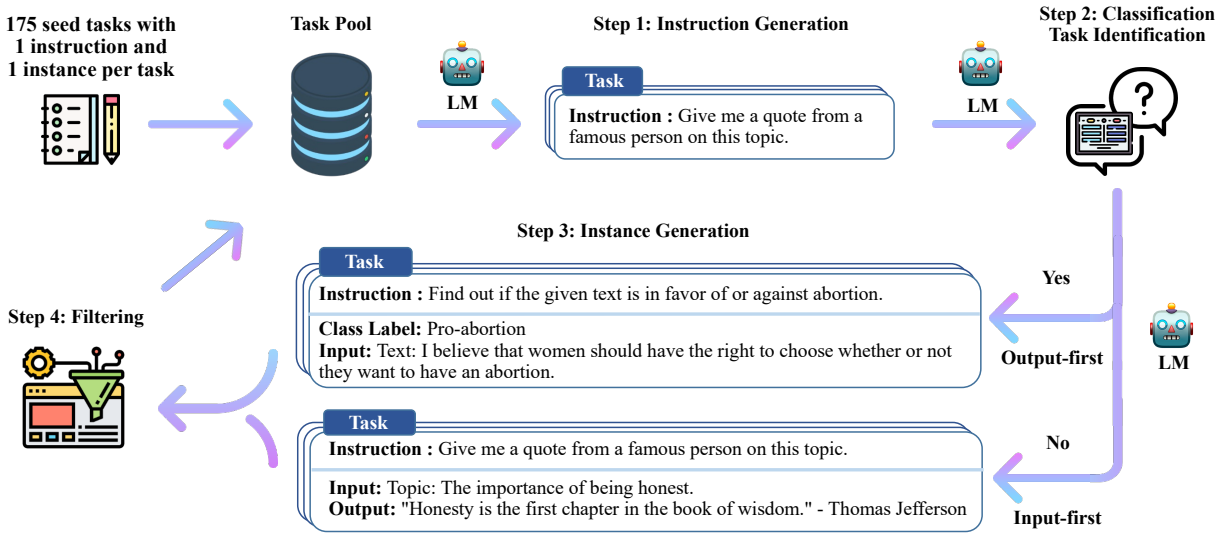


Figure: A high-level overview of SELF-INSTRUCT. The process starts with a small seed set of tasks as the task pool. Random tasks are sampled from the task pool, and used to prompt an off-the-shelf LM to generate both new instructions and corresponding instances, followed by filtering low-quality or similar generations, and then added back to the initial repository of tasks. The resulting data can be used for the instruction tuning of the language model itself later to follow instructions better. Tasks shown in the figure are generated by GPT3.

---

1) Wang, Yizhong, et al. "Self-instruct: Aligning language model with self generated instructions." arXiv preprint arXiv:2212.10560 (2022).

2) Huang, Jiaxin, et al. "Large language models can self-improve." arXiv preprint arXiv:2210.11610 (2022).

3) `https://crfm.stanford.edu/2023/03/13/alpaca.html`

4) Shumailov, Ilia, et al. "The Curse of Recursion: Training on Generated Data Makes Models Forget." arXiv preprint

Open LLMs[1]

- BigScience BLOOM[2]
  - 560M, 1.1B, 1.7B, 3B, 7.1B, 176B
  - BLOOMZ: instruction tuned version
- MosaicML MPT[3]
  - MPT-7B Base (costs $200k), MPT-7B-Instruct, MPT-7B-Chat
  - MPT-7B-StoryWriter-65k+: finetuned from MPT-7B with 65k context size on fiction books, can extrapolate to 84k with ALiBi. (costs $4270, 32xA100-80GB, 2.2 Days, 5B tokens)
- TII Falcon[4]
  - 7B, 40B, 180B
- Pythia, Llama, Mistral, Zephyr, etc.

1) https://github.com/eugeneyan/open-llms
2) https://bigscience.huggingface.co/blog/bloom
3) https://www.mosaicml.com/blog/mpt-7b
4) https://huggingface.co/blog/falcon

Chapter 9

Evaluation

- ChatGPT broke the Turing test — the race is on for new ways to assess AI[1]

Leaderboard | Benchmark | Index

1) https://www.nature.com/articles/d41586-023-02361-7

Open LLM Leaderboard[1]

🤗 Open LLM Leaderboard

📐 With the plethora of large language models (LLMs) and chatbots being released week upon week, often with grandiose claims of their performance, it can be hard to filter out the genuine progress that is being made by the open-source community and which model is the current state of the art. The 🤗 Open LLM Leaderboard aims to track, rank and evaluate LLMs and chatbots as they are released.

🤗 A key advantage of this leaderboard is that anyone from the community can submit a model for automated evaluation on the 🤗 GPU cluster, as long as it is a 🤗 Transformers model with weights on the Hub. We also support evaluation of models with delta-weights for non-commercial licensed models, such as LLaMa.

📈 We evaluate models on 4 key benchmarks from the Eleuther AI Language Model Evaluation Harness , a unified framework to test generative language models on a large number of different evaluation tasks:

- AI2 Reasoning Challenge (25-shot) - a set of grade-school science questions.
- HellaSwag (10-shot) - a test of commonsense inference, which is easy for humans (~95%) but challenging for SOTA models.
- MMLU (5-shot) - a test to measure a text model's multitask accuracy. The test covers 57 tasks including elementary mathematics, US history, computer science, law, and more.
- TruthfulQA (0-shot) - a benchmark to measure whether a language model is truthful in generating answers to questions.

We chose these benchmarks as they test a variety of reasoning and general knowledge across a wide variety of fields in 0-shot and few-shot settings.

Model	Revision	Average ↑	ARC (25-shot) ↑	HellaSwag (10-shot)
tiiuae/falcon-40b-instruct	main	63.2	61.6	84.4

1) https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

MMLU[1,2]

- a massive test consisting of multiple-choice questions for 57 tasks
- subjects: humanities (e.g., law, philosophy, history), social science (e.g., economics, sociology, politics, geography, psychology), STEM (e.g., physics, computer science, mathematic), and other (e.g., medicine, finance, accounting)
- manually collected from freely available sources online
- 15908 questions in total. The few-shot development set has 5 questions per subject, the validation set may be used for selecting hyperparameters and is made of 1540 questions, and the test set has 14079 questions.

Few Shot Prompt and Predicted Answer

The following are multiple choice questions
about high school mathematics.

How many numbers are in the list 25, 26, ..., 100?
(A) 75 (B) 76 (C) 22 (D) 23
Answer: B

Compute $i + i^2 + i^3 + \cdots + i^{258} + i^{259}$.
(A) -1 (B) 1 (C) i (D) -i
Answer: A

If 4 daps = 7 yaps, and 5 yaps = 3 baps,
how many daps equal 42 baps?
(A) 28 (B) 21 (C) 40 (D) 30
Answer: C

1) Hendrycks, Dan, et al. "Measuring massive multitask language understanding." arXiv preprint arXiv:2009.03300 (2020).

2) https://huggingface.co/blog/evaluating-mmlu-leaderboard

Benchmarks

HELM[1,2]

- Multi-metric multi-scenario evaluation of language models under standardized conditions.

Reasoning

- arithmetic reasoning: SingleEq, AddSub, MultiArith, AQUA-RAT, GSM8K, SVAMP
- commonsense reasoning: CommonsenseQA, StrategyQA
- symbolic reasoning: Last Letter Concatenation and Coin Flip
- logical reasoning: Date Understanding, Tracking Shuffled Objects

Language Model Evaluation Harness[3]

- a unified framework to test generative language models on a large number of different evaluation tasks.

---

1) Liang, Percy, et al. "Holistic evaluation of language models." arXiv preprint arXiv:2211.09110 (2022).
2) `https://crfm.stanford.edu/helm/latest/`
3) `https://github.com/EleutherAI/lm-evaluation-harness`

The Foundation Model Transparency Index[1]

Foundation Model Transparency Index Scores by Major Dimensions of Transparency, 2023

Source: 2023 Foundation Model Transparency Index

Major Dimensions of Transparency	Meta Llama 2	BigScience BLOOMZ	OpenAI GPT-4	stability.ai Stable Diffusion 2	Google PaLM 2	ANTHROPIC Claude 2	cohere Command	AI21labs Jurassic-2	Inflection Inflection-1	amazon Titan Text	Average
Data	40%	60%	20%	40%	20%	0%	20%	0%	0%	0%	**20%**
Labor	29%	86%	14%	14%	0%	29%	0%	0%	0%	0%	**17%**
Compute	57%	14%	14%	57%	14%	0%	14%	0%	0%	0%	**17%**
Methods	75%	100%	50%	100%	75%	75%	0%	0%	0%	0%	**48%**
Model Basics	100%	100%	50%	83%	67%	67%	50%	33%	50%	33%	**63%**
Model Access	100%	100%	67%	100%	33%	33%	67%	33%	0%	33%	**57%**
Capabilities	60%	80%	100%	40%	80%	80%	60%	60%	40%	20%	**62%**
Risks	57%	0%	57%	14%	29%	29%	29%	29%	0%	0%	**24%**
Mitigations	60%	0%	60%	0%	40%	40%	20%	0%	20%	20%	**26%**
Distribution	71%	71%	57%	71%	71%	57%	57%	43%	43%	43%	**59%**
Usage Policy	40%	20%	80%	40%	60%	60%	40%	20%	60%	20%	**44%**
Feedback	33%	33%	33%	33%	33%	33%	33%	33%	33%	0%	**30%**
Impact	14%	14%	14%	14%	14%	0%	14%	14%	14%	0%	**11%**
Average	**57%**	**52%**	**47%**	**47%**	**41%**	**39%**	**31%**	**20%**	**20%**	**13%**	

1) https://crfm.stanford.edu/fmti/

Chapter 10

Engineering

- Real-world practice.

P2P | Parallelism | Libraries

Petals

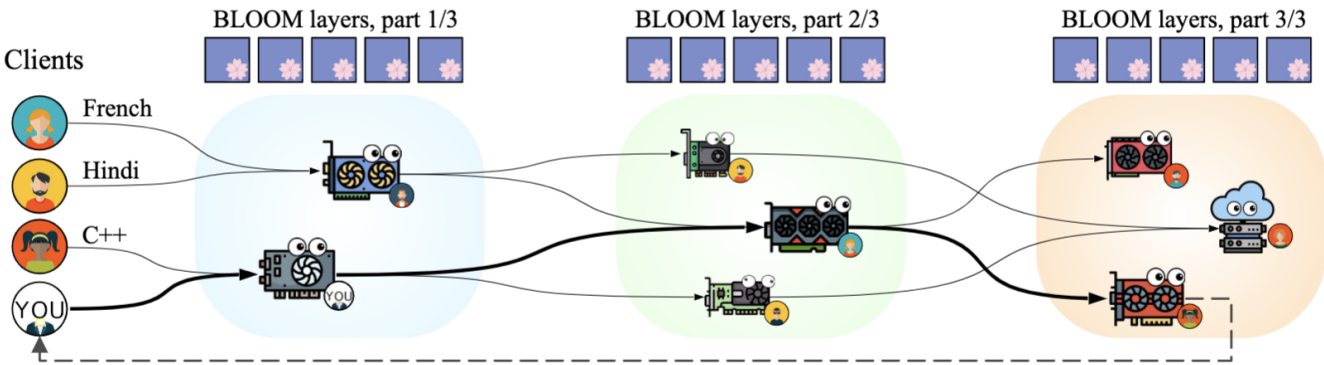


Figure 1: An overview of PETALS. Some participants (*clients*) want to use a pretrained language model to solve various tasks involving processing texts in natural (e.g., French, Hindi) or programming (e.g., C++) languages. They do it with help of other participants (*servers*), who hold various subsets of model layers on their GPUs. Each client chooses a sequence of servers so that it performs an inference or fine-tuning step in the least amount of time.

Petals[1,2] runs large language models like BLOOM-176B (over 350 GB) **collaboratively** - you load a small part (8 GB) of the model, then team up with people serving the other parts to run inference or fine-tuning.
Inference runs at around **1 sec per step (token)** - 10x faster than possible with offloading, enough for chatbots and other interactive apps. Parallel inference reaches hundreds of tokens/sec.

---

1) `https://github.com/bigscience-workshop/petals`

2) Borzunov, Alexander, et al. "Petals: Collaborative inference and fine-tuning of large models." arXiv preprint arXiv:2209.01188 (2022).

3D parallelism training

Distributed training, 3D parallelism of BLOOM.[1,2]

- **Data parallelism** (DP): replicates the model multiple times, with each replica placed on a different device and fed a slice of the data. The processing is done in parallel and all model replicas are synchronized at the end of each training step.
  - Distributed Data Parallel (DDP), Fully Sharded Data Parallel (**FSDP**)[3]
- **Tensor parallelism** (TP): partitions individual layers of the model across multiple devices. This way, instead of having the whole activation or gradient tensor reside on a single GPU, we place shards of this tensor on separate GPUs. This technique is sometimes called horizontal parallelism or intra-layer model parallelism.
- **Pipeline parallelism** (PP): splits up the model's layers across multiple GPUs, so that only a fraction of the layers of the model are placed on each GPU. This is sometimes called vertical parallelism.

1) `https://huggingface.co/blog/bloom-megatron-deepspeed`
2) `https://github.com/bigscience-workshop/Megatron-DeepSpeed`
3) `https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html`

3D parallelism training

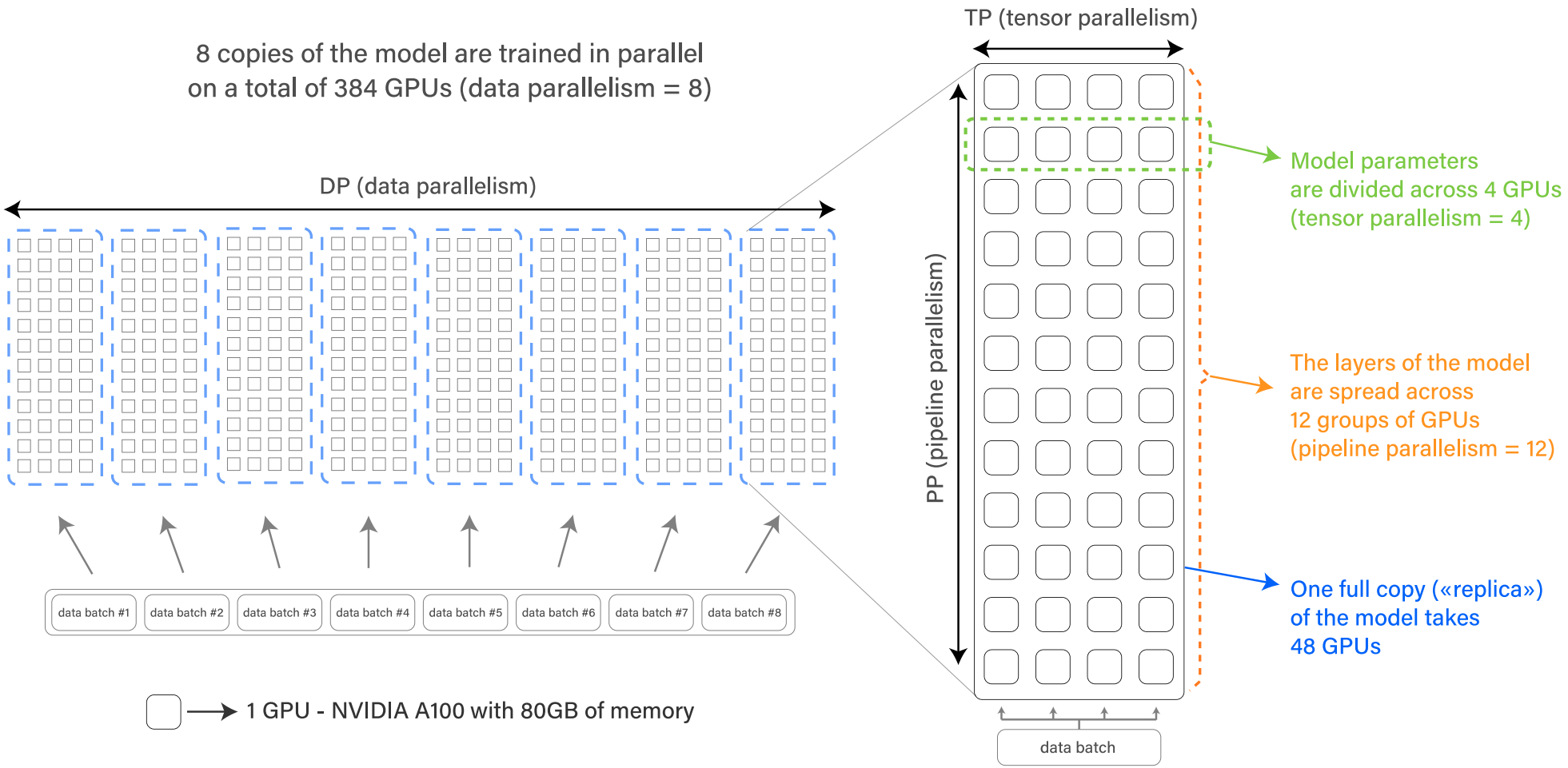


Figure: DP+PP+TP combination leads to 3D parallelism.

Python libraries

- **Transformers**: transformer framework, Hugging Face.
- **Accelerate**: enables running PyTorch training scripts in a distributed environment (single CPU, single GPU, multi-GPUs and TPUs).

- **Lit-GPT**: transformer framework, Lightning AI.
- **Lightning** Fabric: scales PyTorch models on distributed machines while maintaining full control of your training loop.

- **Megatron-LM**: transformer framework, NVIDIA.
- **DeepSpeed**: a deep learning optimization library that makes distributed training easy, efficient, and effective.

- **Text Generation Inference**: Fast optimized inference for LLMs.
- **GGML**: Fast CPU Inference of LLMs in pure C/C++.
- **Intel Extension for Transformers**: run LLMs efficiently on Intel CPU Platforms.
- **vLLM**: an open-source library for fast LLM inference and serving, with PagedAttention algorithm.

Chapter 11

Future directions

- What's next?

Multi-modality

MoE

RAG

Multi-modality

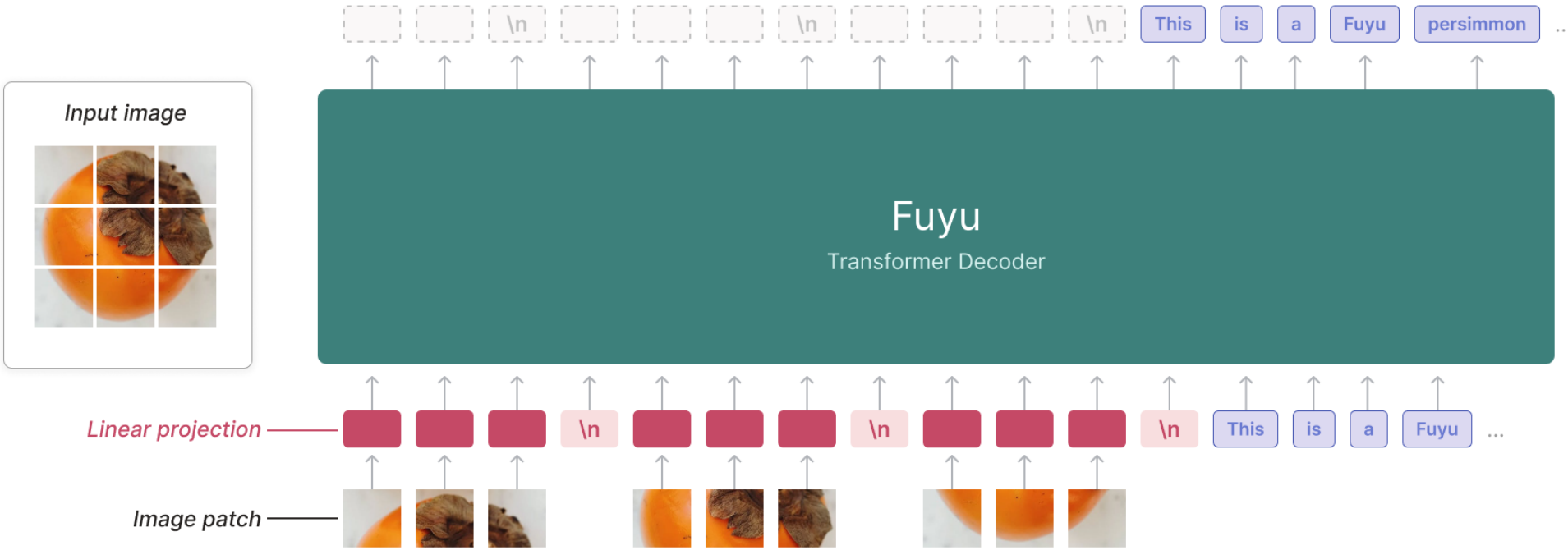


Figure: Fuyu-8B, decoder-only, visual question-answering and image-captioning.

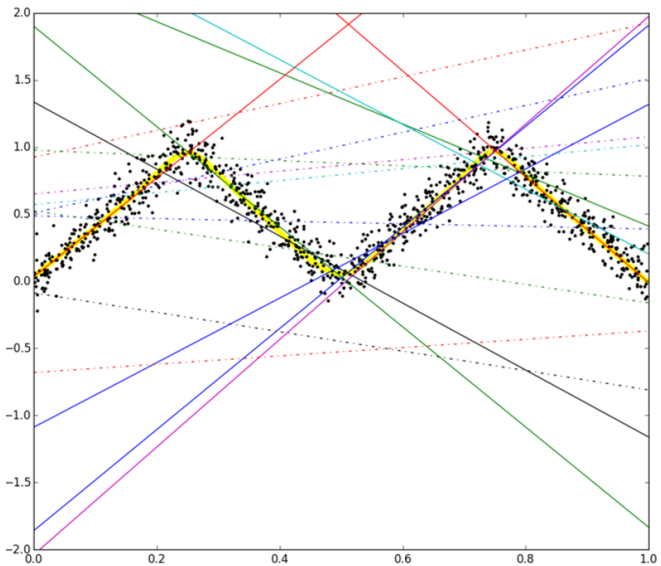
Image patches are linearly projected directly into the first layer of the transformer, bypassing the embedding lookup.

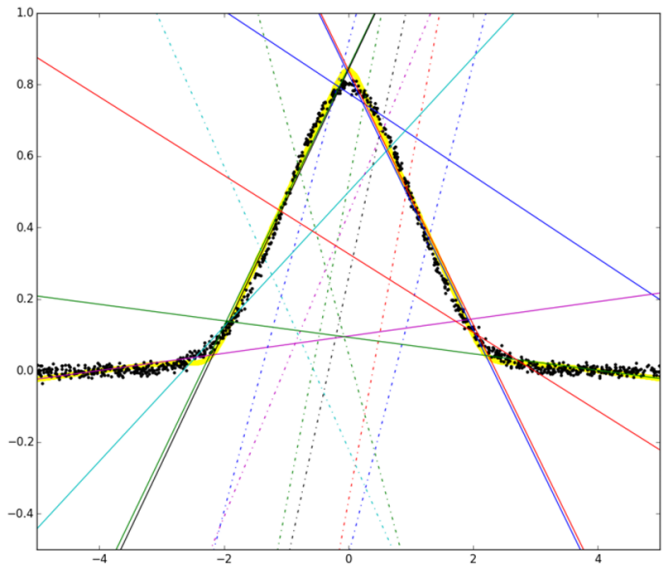
1) `https://www.adept.ai/blog/fuyu-8b`

Mixture-of-Experts (MoE)[1]

$$p(t|x) = \Sigma_{k=1}^{K} \pi_k(x) p_k(t|x)$$

where the mixing coefficients $\pi_k(x)$ are known as gating functions and the individual component densities $p_k(t|x)$ are called experts. An example with linear experts:

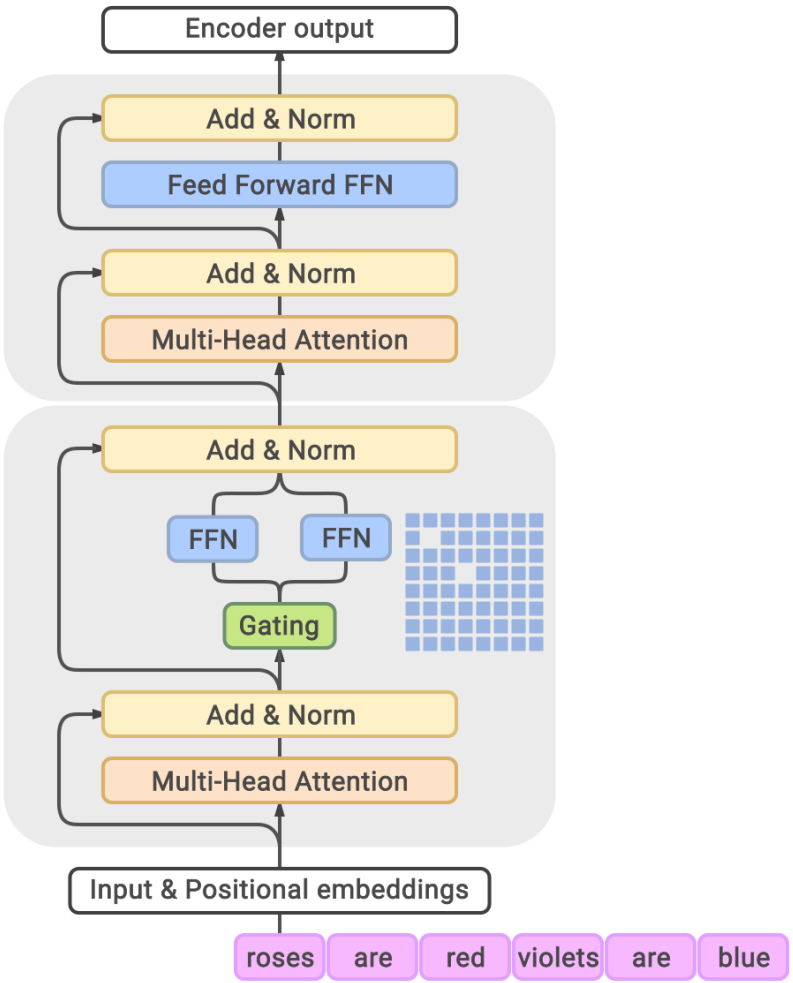




The notion behind the terminology is that different components can model the distribution in different regions of input space (they are "experts" at making predictions in their own regions), and the gating functions determine which components are dominant in which region.

1) Bishop, Christopher M., and Nasser M. Nasrabadi. Pattern recognition and machine learning. Vol. 4. No. 4. New York: springer, 2006.

Mixture-of-Experts (MoE)



A single giant (1T+ parameters) model to represent many (e.g., 64) expert sub-models.

a subset of experts is selected on a per-token or per-example basis $\Rightarrow$ only corresponding parts of the network are activated $\Rightarrow$ sparse, reducing computational cost

Figure: GLaM model architecture. Each MoE layer (the bottom block) is interleaved with a Transformer layer (the upper block). For each input token, e.g., 'roses', the *Gating* module dynamically selects two most relevant experts out of 64, which is represented by the blue grid in the MoE layer. The weighted average of the outputs from these two experts will then be passed to the upper Transformer layer. For the next token in the input sequence, two different experts will be selected.

1) Du, Nan, et al. "Glam: Efficient scaling of language models with mixture-of-experts." International Conference on Machine Learning. PMLR, 2022.

2) `https://www.forbes.com/sites/robtoews/2023/02/07/the-next-generation-of-large-language-models`

Retrieval Augmented Generation (RAG)

Assumption: LLMs encoded everything they know in their model parameters. While this makes sense for language information, it is inefficient for factual and world-knowledge information.
⇒ separate language modeling (LLM) and world knowledge information (database)
⇒ makes LLM smaller + deals with **hallucinations**

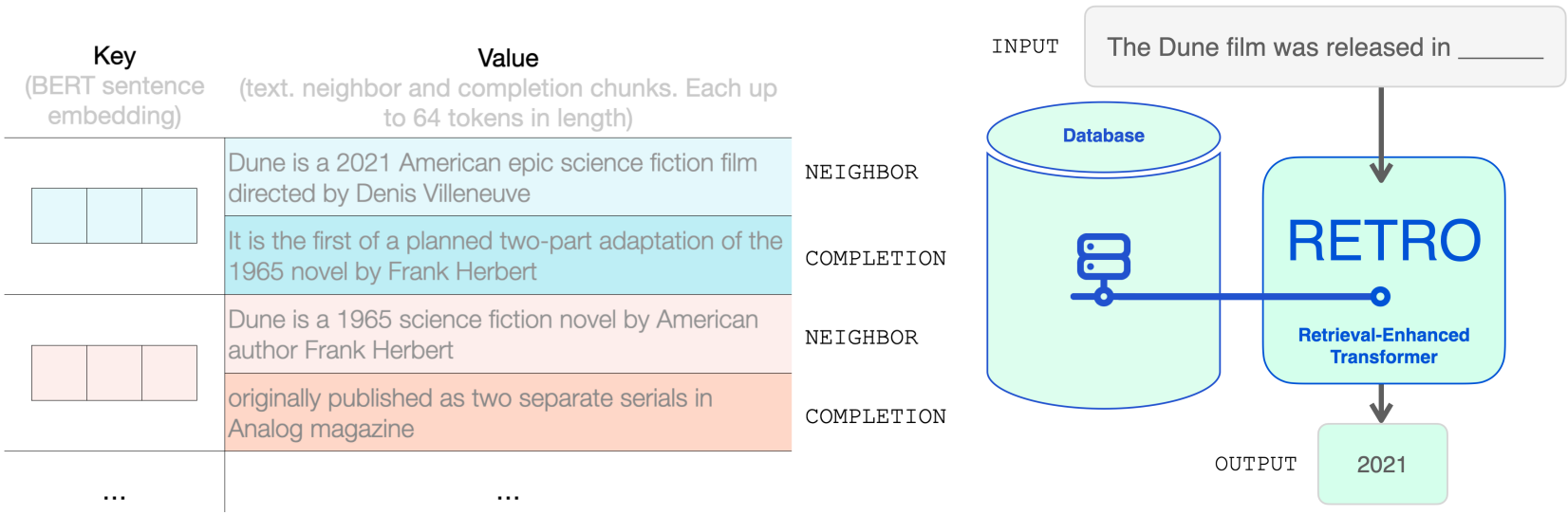


Figure: RETRO: Retrieval-Enhanced Transformer[1,2]

1) Borgeaud, Sebastian, et al. "Improving language models by retrieving from trillions of tokens." International conference on machine learning. PMLR, 2022.

2) `https://jalammar.github.io/illustrated-retrieval-transformer/`