CSE 585 Post-Training

October 2, 2024

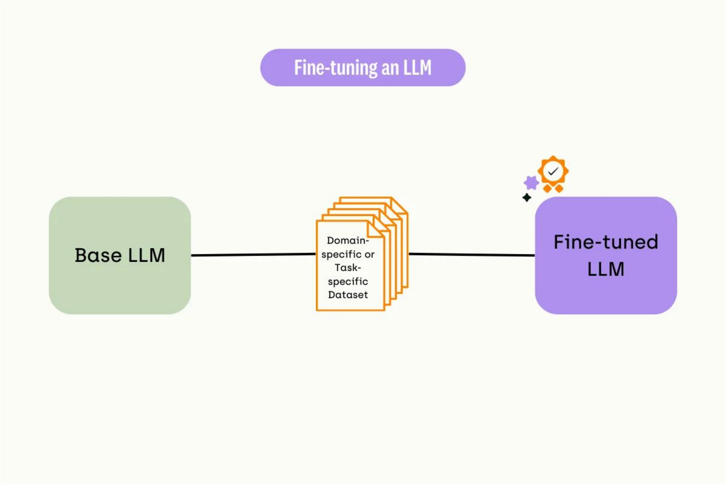
Oskar Shiomi Jensen, Conor Wilkinson, Kevin Sun

LoRa: Low-Rank Adaptation of Large Language Models

Edward Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu,
Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen

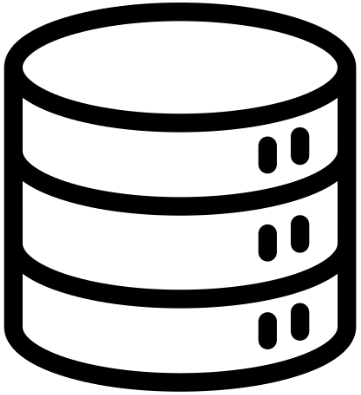
Background: What is Post-Training?

- Typically trains model on general domain data (pre-training)
- Post-training adapts general model to particular tasks or domains
    - Full Fine-tuning: Update All Parameters of the model

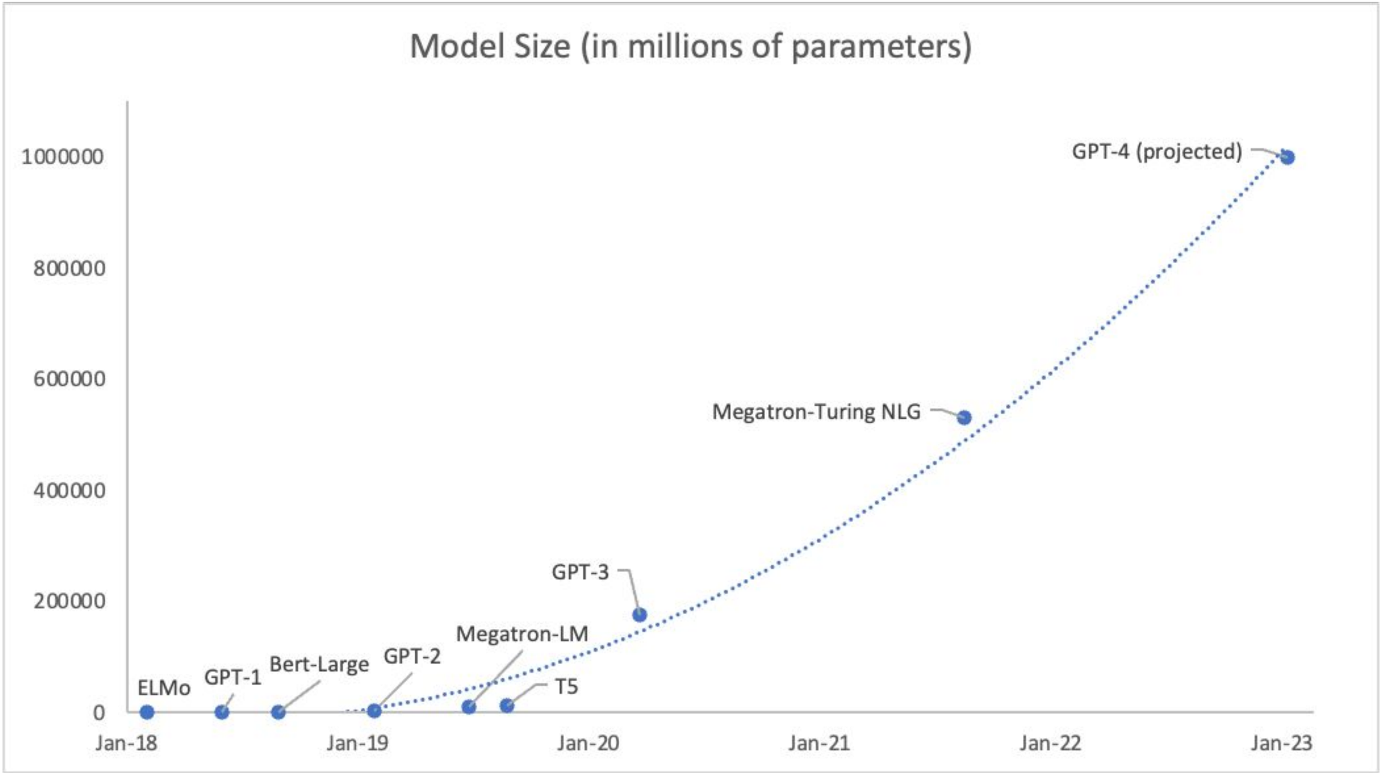


Why Not Full Fine-Tuning?

- Storage overhead
    - GPT 3 has 175 B parameters
    - More than 350 GB for each adaptation!
    - Checkpoints = model weights/biases + optimizer states
- Swapping between fine-tuned models becomes non-trivial due to size

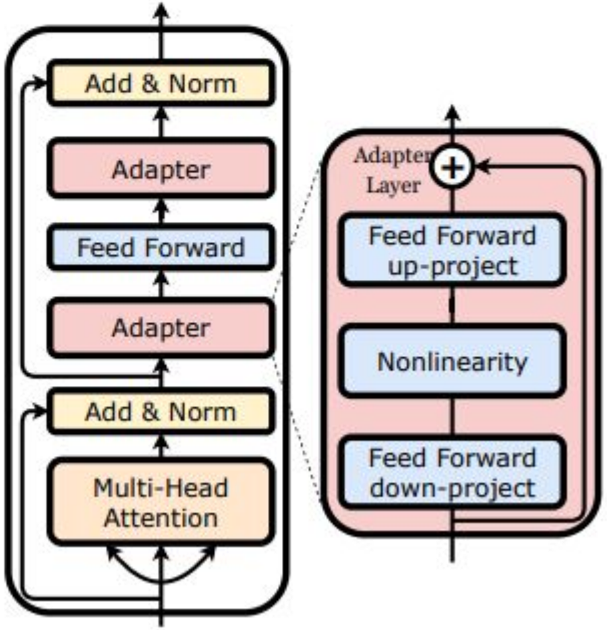


Background: LLM Parameter Growth



Existing Solutions: Adapter Layers

- Train additional modules in transformer
- Increased latency due to increased depth of the transformer (+20-30%)

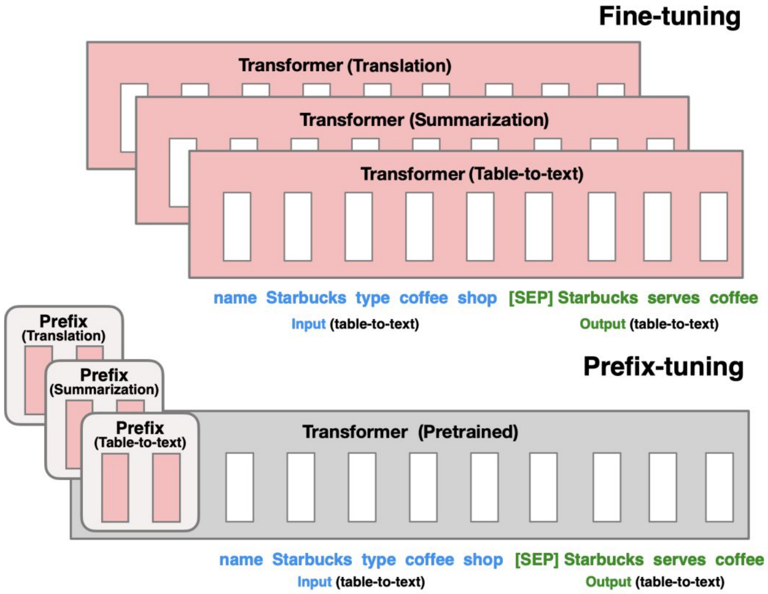


Existing Solutions: Direct Optimization of Prompt

Prefix tuning: Adding special characters (word embeddings) into the prompt

- Performance changes non-monotonically in trainable parameters
- Reduces available sequence length

Generally underperforms



Background: Matrix Rank

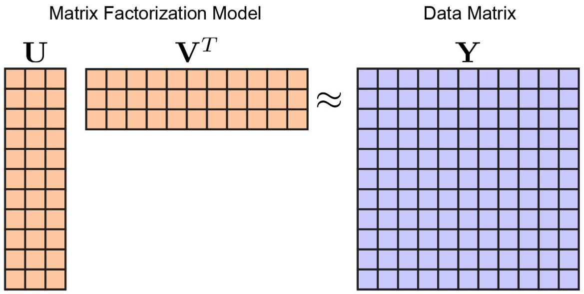
Rank = number of linearly independent rows or columns in a matrix

$$rank(A) \leq min(m,n)$$

Linearly independent rows and columns encode information into the matrix

Background: Matrix Factorization

Matrices can be approximated by the matrix multiplication of two lower rank matrices.

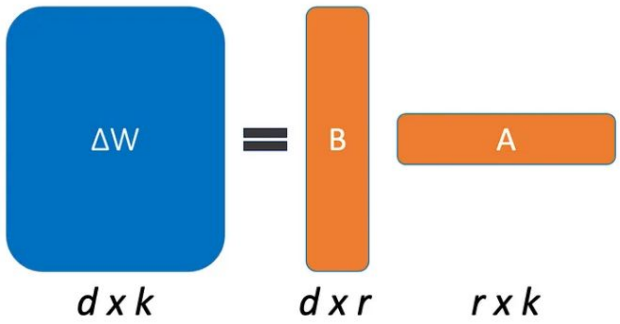


For example: PCA decomposition

Background: Low Rank Structures in Deep Learning

Measuring the Intrinsic Dimension of Objective Landscapes [Li et al.]

- Found models are overparameterized
- Less parameters can be used for same level of performance
- Model matrices can be decomposed into matrices of lower rank

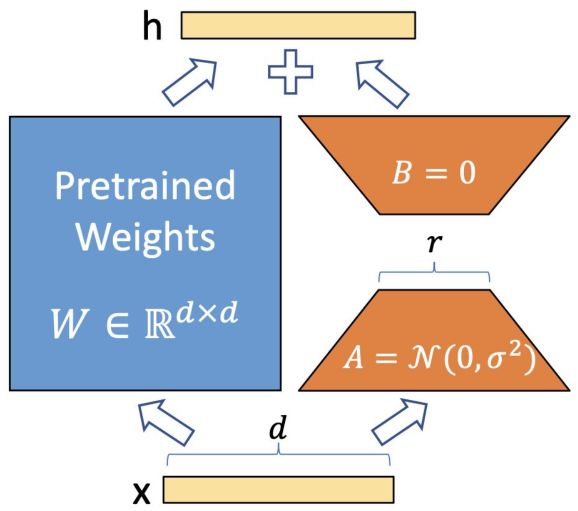


Solution: Low-Rank Adaptation (LoRA)

Hypothesized ΔW has low "intrinsic rank"

$$B \in \mathbb{R}^{d \times r}$$

$$A \in \mathbb{R}^{r \times k}$$



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

Solution: Low-Rank Adaptation (LoRA)

Memory and storage efficiency

Parameters go from d^2 to 2dr

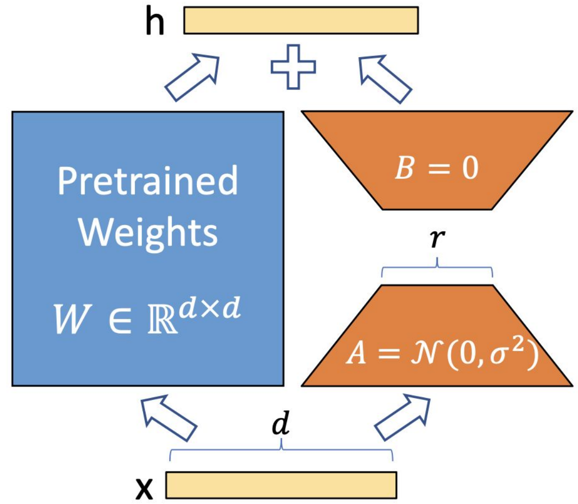
LoRA Implementation

Freeze the pretrained weights, W

B initialized to 0, A initialized to a Gaussian distribution

Make updates on B and A

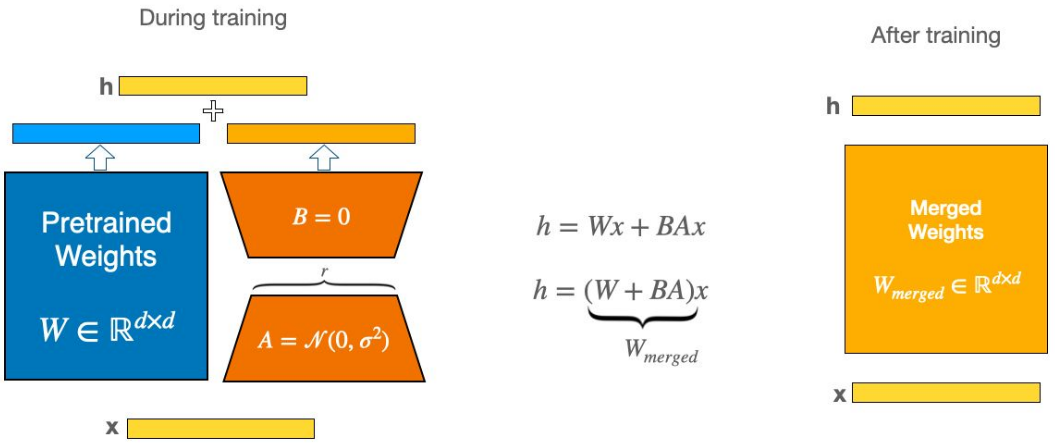
Hyperparameter $\frac{\alpha}{r}$ is used to scale BAx much like a learning rate



Advantages of LoRA

No Additional Inference Latency

Can switch to different task models more quickly



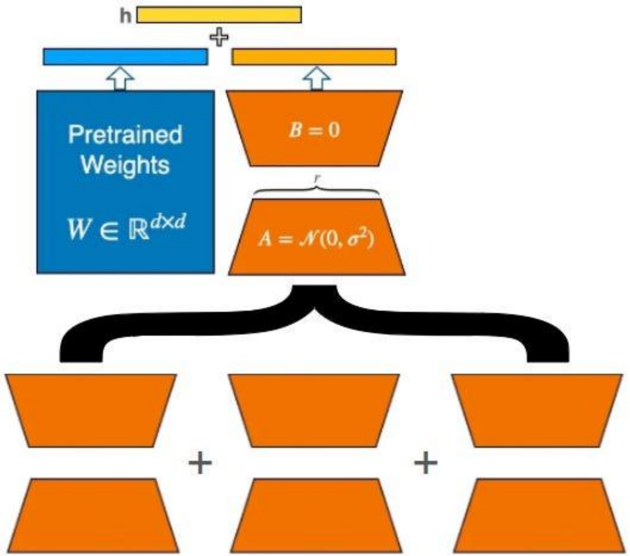
Advantages of LoRA

Generalization of Full Fine-Tuning

- We can change r
- As r increases, it converges to full fine-tuning

Limitations

Not straightforward to batch input to different tasks with different A and B in a single forward pass, if A and B is absorbed into W.



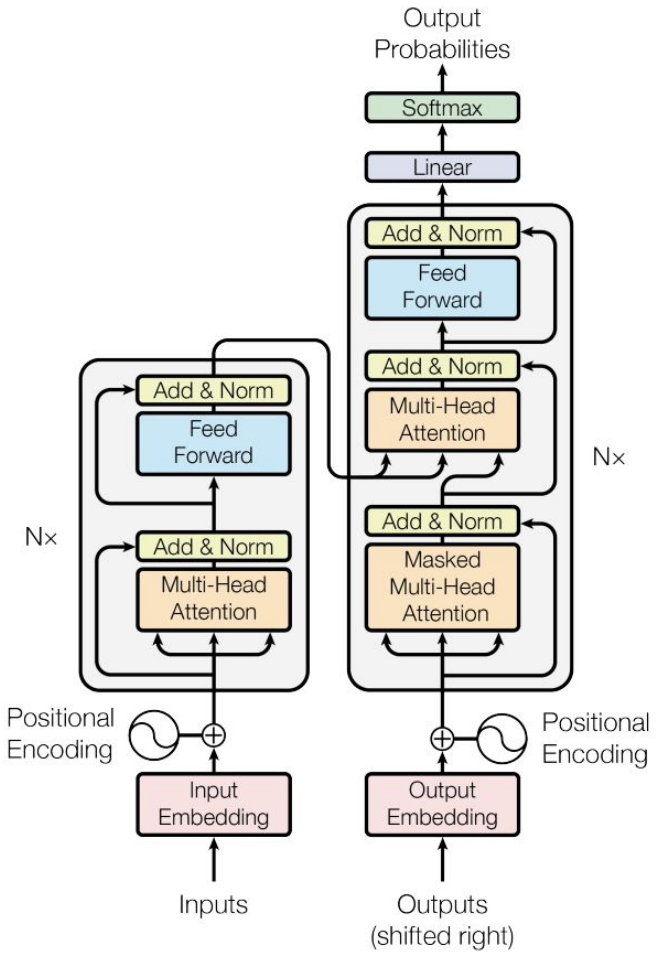
Applying LoRA to GPT-3

Only adapted the attention weights

$$W_q, W_k, W_v, W_o$$

Treat Wq, Wk, and Wv as a single matrix

Freeze MLP modules

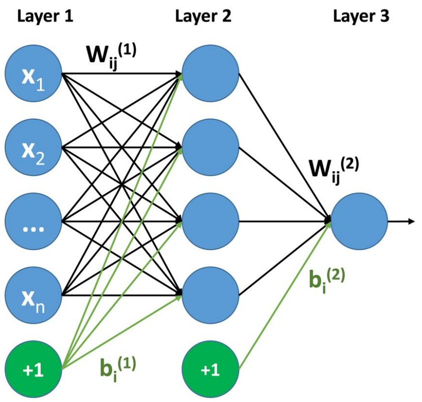


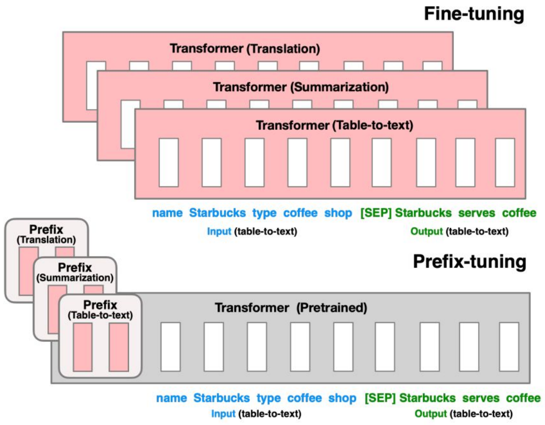
How does it affect the system?

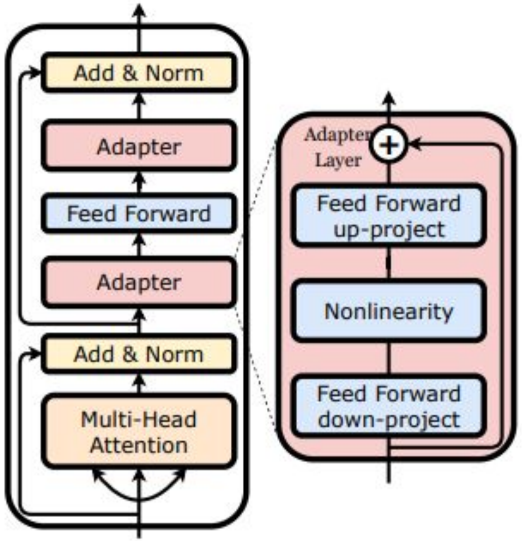
GPT-3 175B with rank 4 and tuning only query and value matrices

- VRAM consumption 1.2TB -> 350GB
  - No optimizer states for frozen parameters with Adam
  - No gradients for frozen parameters
- Checkpoint size 350GB -> 35MB (10,000x decrease)
- 25% speedup during fine-tuning
  - No need to calculate gradients for most parameters

Baseline Methods for Evaluation







Bias-Only

Prefix Embedding

Adapter Layer

Finding Optimal Parameters

Set parameter budget, then grid search through combinations of rank and subsets of matrices to adapt.

	# of Trainable Parameters = 18M						
Weight Type Rank r	W_q 8	W_k 8	W_v 8	W_o 8	W_q, W_k 4	W_q, W_v 4	W_q, W_k, W_v, W_o 2
WikiSQL ($\pm 0.5\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Evaluation

- RoBERTa, GPT-2, GPT-3 benchmarks
- LoRA generally outperforms all other benchmarks, even full fine-tuning, while only training a small fraction of parameters.
- LoRA does well at common natural language tasks like summarization and Q&A, and also more complex problems such as language to SQL.

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB$_{base}$ (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB$_{base}$ (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB$_{base}$ (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm}$[illegible]	84.4
RoB$_{base}$ (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm}$[illegible]	85.4
RoB$_{base}$ (LoRA)	0.3M	$87.5_{\pm.3}$	**95.1**$_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	**93.3**$_{\pm.3}$	$90.8_{\pm.1}$	**86.6**$_{\pm.7}$	**91.5**$_{\pm}$[illegible]	**87.2**
RoB$_{large}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
RoB$_{large}$ (LoRA)	0.8M	**90.6**$_{\pm.2}$	$96.2_{\pm.5}$	**90.9**$_{\pm1.2}$	**68.2**$_{\pm1.9}$	**94.9**$_{\pm.3}$	$91.6_{\pm.1}$	**87.4**$_{\pm2.5}$	**92.6**$_{\pm}$[illegible]	**89.0**
RoB$_{large}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	**68.3**$_{\pm1.0}$	**94.8**$_{\pm.2}$	**91.9**$_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm}$[illegible]	88.4
RoB$_{large}$ (Adpt^{P})†	0.8M	**90.5**$_{\pm.3}$	**96.6**$_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	**94.8**$_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm}$[illegible]	87.9
RoB$_{large}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.}$[illegible]	87.8
RoB$_{large}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm}$[illegible]	86.4
RoB$_{large}$ (LoRA)†	0.8M	**90.6**$_{\pm.2}$	$96.2_{\pm.5}$	**90.2**$_{\pm1.0}$	$68.2_{\pm1.9}$	**94.8**$_{\pm.3}$	$91.6_{\pm.2}$	**85.2**$_{\pm1.1}$	**92.3**$_{\pm}$[illegible]	**88.6**
DeB$_{XXL}$ (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB$_{XXL}$ (LoRA)	4.7M	**91.9**$_{\pm.2}$	$96.9_{\pm.2}$	**92.6**$_{\pm.6}$	**72.4**$_{\pm1.1}$	**96.0**$_{\pm.1}$	**92.9**$_{\pm.1}$	**94.9**$_{\pm.4}$	**93.0**$_{\pm}$[illegible]	**91.3**

Model & Method	# Trainable Parameters	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M (Adapter^{L})*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter^{L})*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter^{H})	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	**70.4**$_{\pm.1}$	**8.85**$_{\pm.02}$	**46.8**$_{\pm.2}$	**71.8**$_{\pm.1}$	**2.53**$_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter^{L})	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	**2.49**$_{\pm.0}$
GPT-2 L (Adapter^{L})	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	**70.4**$_{\pm.1}$	**8.89**$_{\pm.02}$	**46.8**$_{\pm.2}$	**72.0**$_{\pm.2}$	$2.47_{\pm.02}$

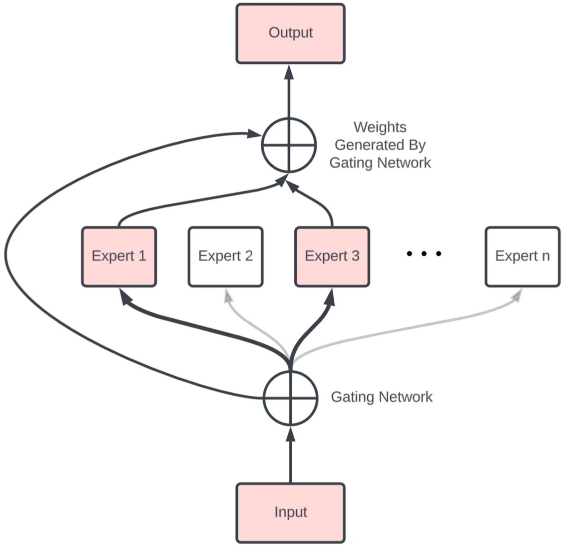
Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter^{H})	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^{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

Sparse Upcycling: Training Mixture of Experts from Dense Checkpoints

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp,
Carlos Riquelme, Basil Mustafa, Joshua Ainslie
Yi Tay, Mostafa Dehghani, Neil Houlsby

Problem Statement

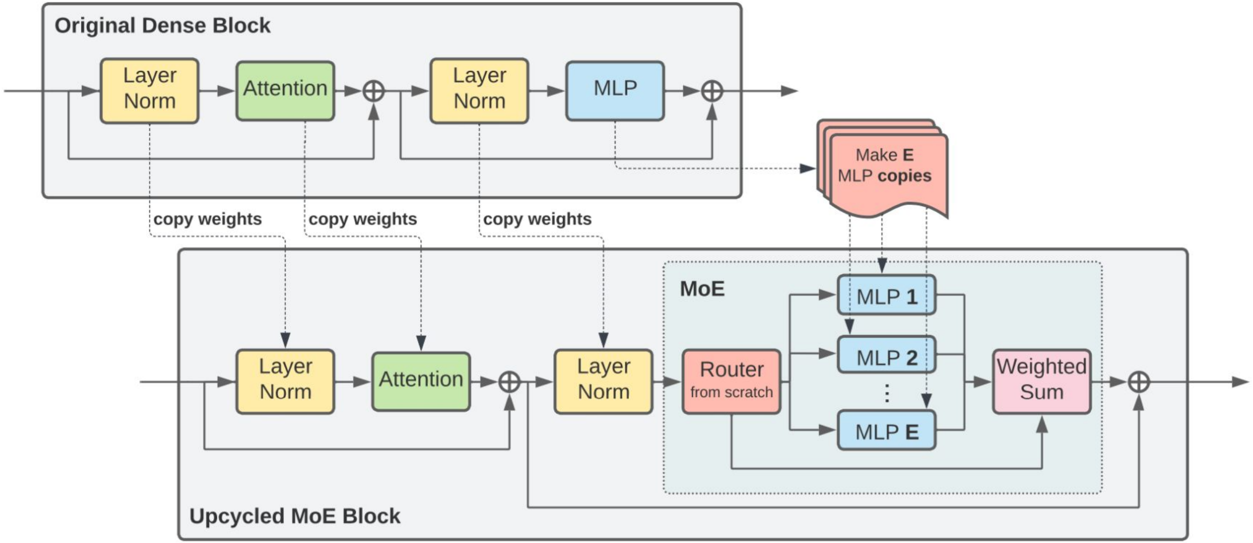
- Mixture of experts (MoE) models perform well by scaling parameters while keeping inference compute relatively low.
- Training MoE models from scratch is extremely expensive.
- We require “model surgery”: a way to adapt what has been learned by dense model to MoE architecture.



Solution: Upcycling

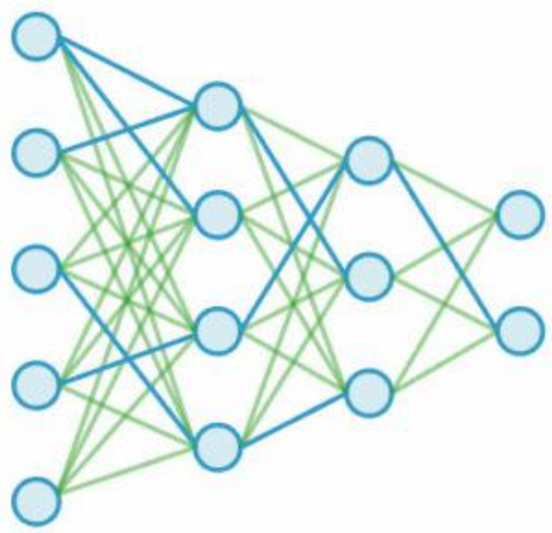
- Already trained dense models can be “upcycled” to make use of the sunk cost of training, which can be 2000+ ZFLOPS (PaLM).
- We can upcycle by converting a subset of dense blocks to MoE blocks
- To do the conversion, insert a routing network and copy dense MLP parameters for each added expert.
- Insight: We are now training with more parameters than dense model, but **begin much closer to convergence.**

Upcycling Visualization



What is currently done to reach high model capacity?

- Dense models apply all parameters to every input
- training dense models to convergence is extremely computationally expensive
- this results in a limited number of dense models to be reused across many different tasks



- sparse upcycling can reach better performance on vision and language tasks with less extra pretraining time

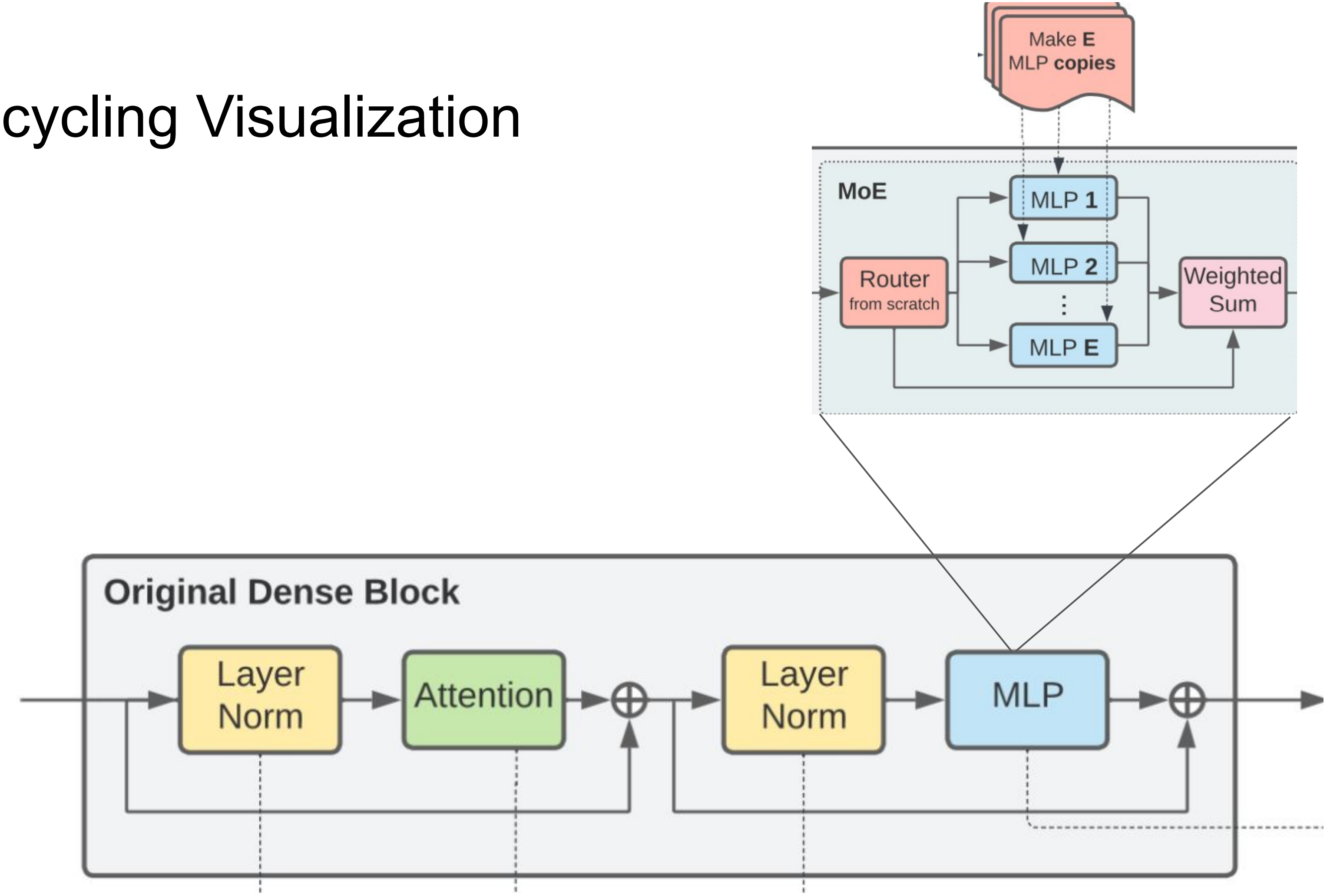
Sparse Upcycling Initialization

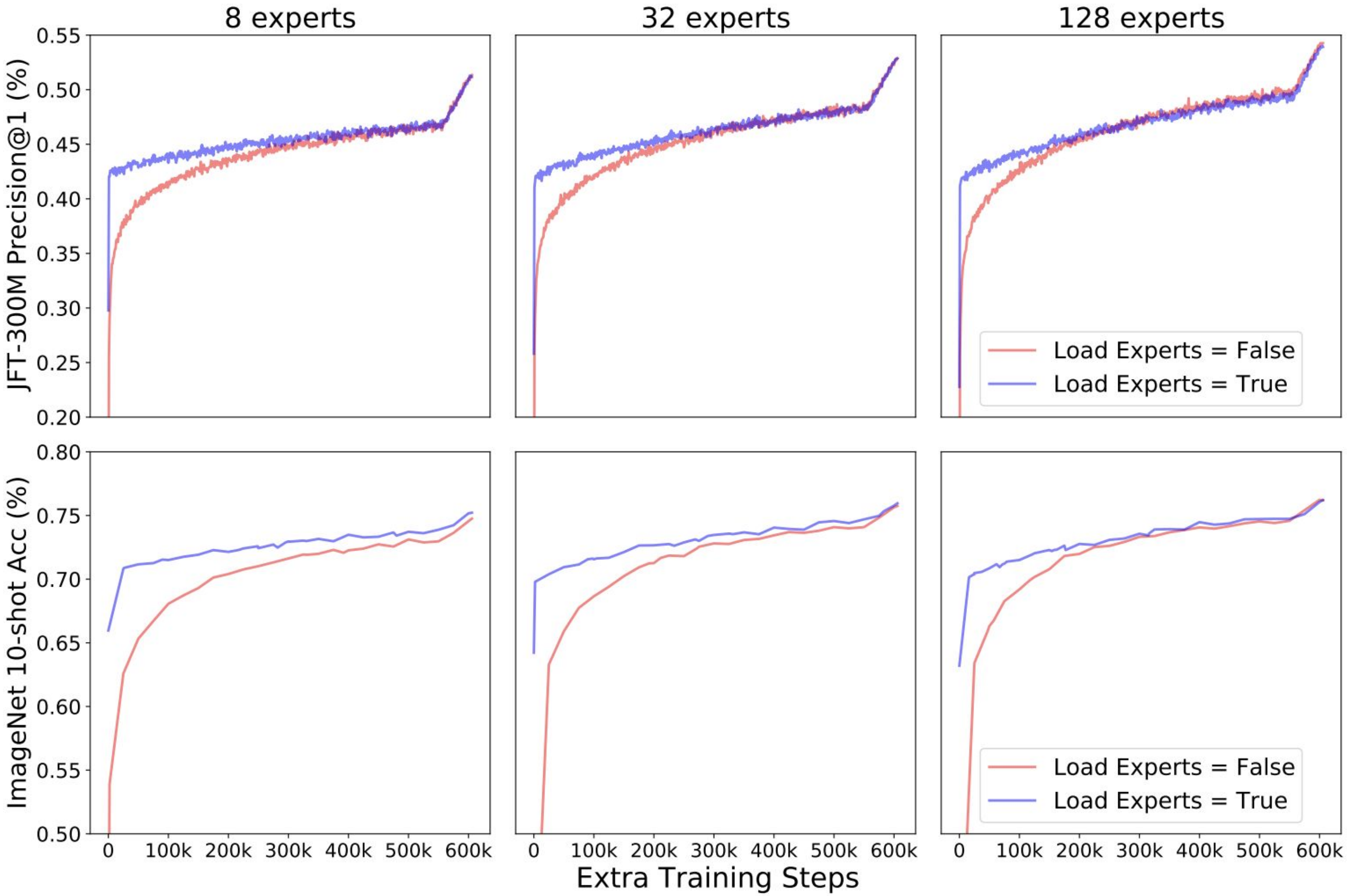
- Sparse upcycling takes advantage of an existing model and upgrades it with low extra computation budget

- all parameters of the original model's training checkpoint are copied
- the experts in the new MoE layer are initialized to be copies of the original MLP layer

- due to changing the trained network's structure, upcycling causes an initial performance decrease

Upcycling Visualization





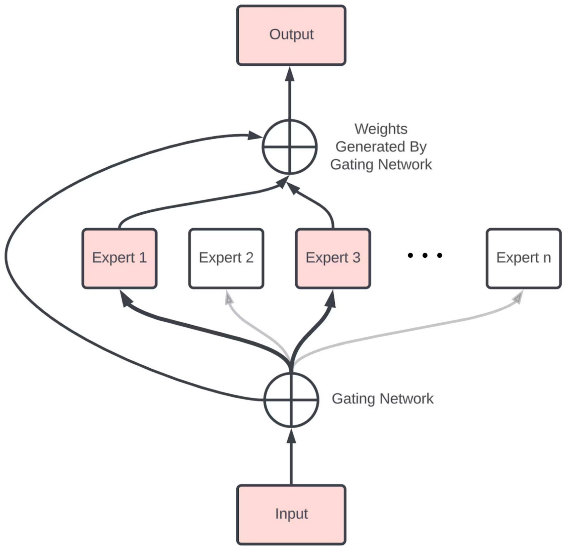
8 experts
32 experts
128 experts
JFT-300M Precision@1 (%)
ImageNet 10-shot Acc (%)
Load Experts = False
Load Experts = True
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.50
0.55
0.60
0.65
0.70
0.75
0.80
0
100k
200k
300k
400k
500k
600k
Extra Training Steps

Routing Methods

- The router is an algorithm that decides which expert is applied to each individual token

- Routing network outputs a probability for each combination of token and expert.

- 2 types of routing methods explored: expert choice and Top-K



Expert Choice

- Let:
  - n = the total number of tokens
  - E = the total number of experts in an MoE layer
- Every expert e chooses the top T tokens with highest probabilities for that expert
- T is parameterized as $T = C(n/E)$ where C is the expert capacity hyperparameter
- C is a controllable variable that allows us to choose more or less tokens per expert
- changing C allows tradeoffs between performance and compute cost.

Top-K

- in Top-K routing, each token is sent to the K experts with highest probability

- Expert Choice routing is used for Vision models and the encoder of Language models

- Top K routing is used for the decoder of Language models

Design Decisions

Router Type:

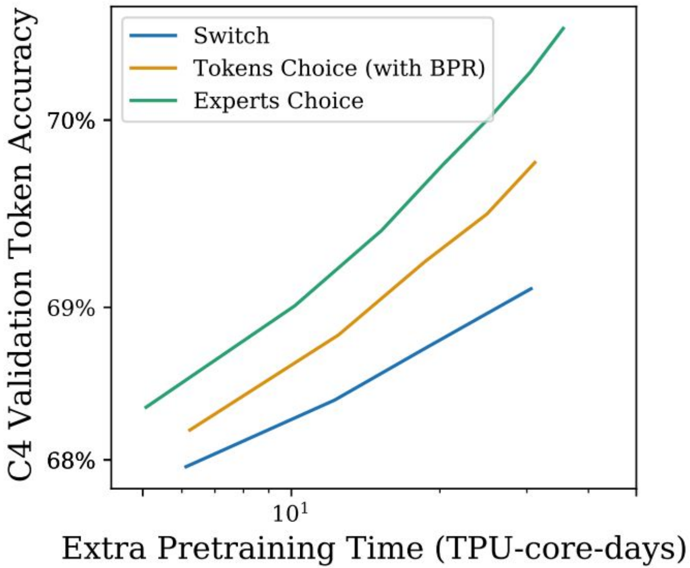
- Vision models use Expert Choice Routing (C=2)
- Language models use Expert Choice Routing (C=2) for the encoder and Top-K Routing (K=2) in the decoder
- C = 2 and K = 2 outperforms C = 1 and K = 1 and all of them outperform the dense continuation model

Model	Capacity	From	Extra Epochs	Val Prec@1	ImageNet 10shot
Dense	–	Dense	7	49.60	73.59
Expert Choice	$C = 1$	Dense	7	51.91	74.04
Top-K	$K = 1$	Dense	7	51.51	74.40
Expert Choice	$C = 2$	Dense	7	52.80	74.83
Top-K	$K = 2$	Dense	7	52.88	74.91

Design Decisions

Router Type:

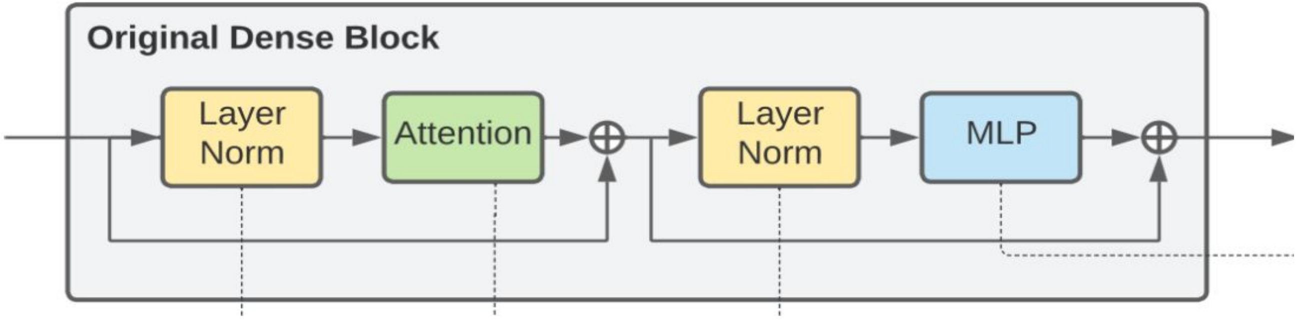
- For Language, Expert Choice routing outperforms both Top-2 routing and Top-1 routing (switch) given the same amount of training time



Design Decisions

Number of Layers to Upcycle:

- replacing more MLP layers with MoE layers:
  - increases model capacity
  - increases the model cost
  - increases initial quality drop from original dense model
- in this paper, MLP layers are replaced with MoE layers consecutively starting from the end of the model

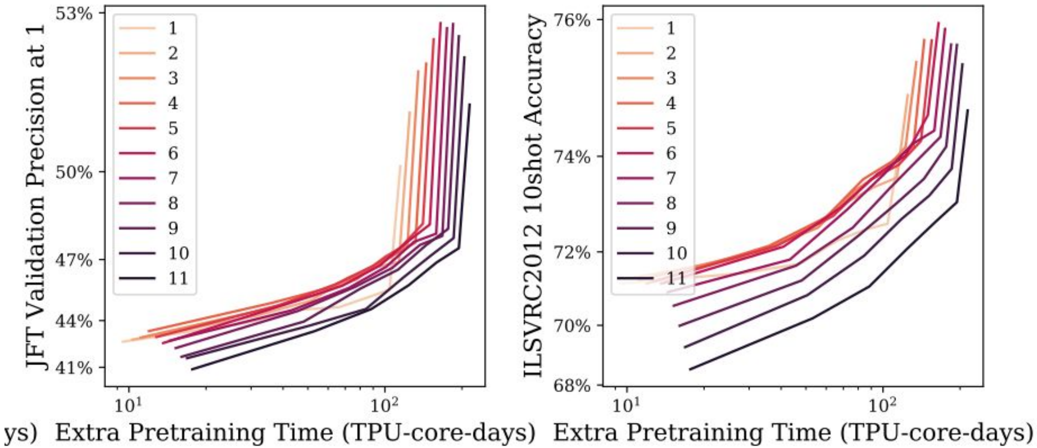


Design Decisions

Number of Layers to Upcycle:

- From the vision experiment plots, we see that for a given additional pre-training time, the best accuracy comes from using 5-6 MoE layers

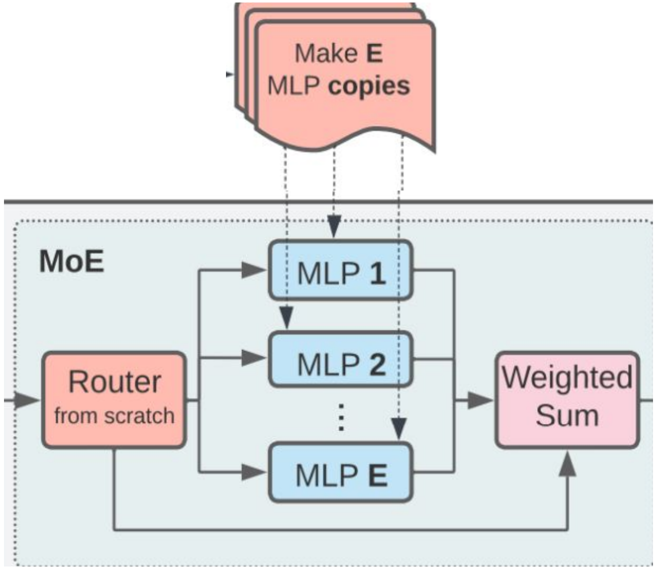
- There are 12 total MLP layers to choose from and experiments are fixed with 32 experts



Design Decisions

Number of Experts to Add in Upcycled Layers

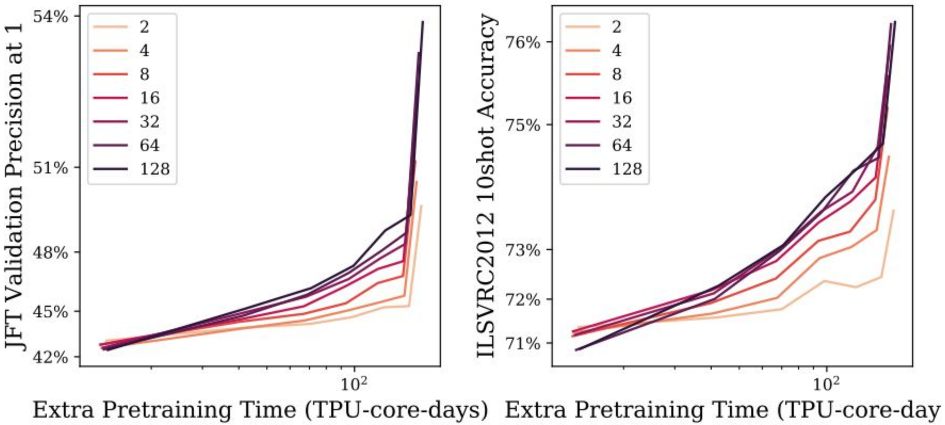
- Increasing the number of experts E used per MoE layer:
  - increases the number of model parameters
  - increases quality of the model (to a point)
  - increases the initial model quality drop
  - does not increase the computation required (due to T being inversely proportional to E)



Design Decisions

Number of Experts to Add in Upcycled Layers

- For the vision experiment, using a fixed amount of extra pretraining time, more experts per MoE layer results in higher performance
- Experiment uses 6 MoE layers with experts ranging from 2-128



Design Decisions

Expert Capacity

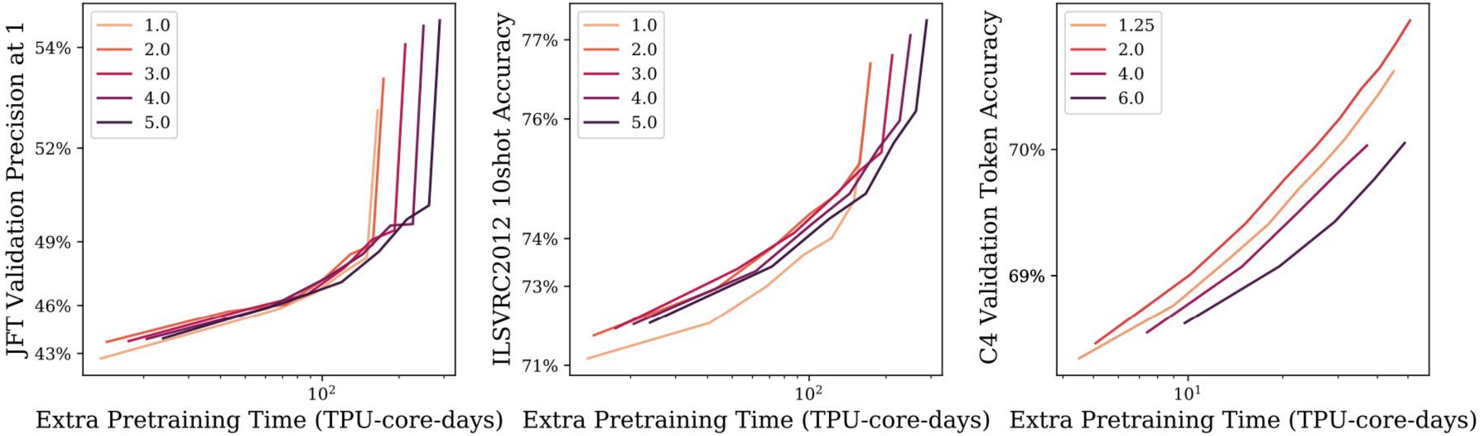
$$T = C * (n/E)$$

- expert capacity C increases the number of tokens selected by each expert
- consequently, the number of experts that process each token also increases
- larger expert capacity
  - increases quality
  - increases FLOPS and run time
- with C = 1, FLOPS is very similar to original dense model
- C = 2 offers good quality on a compute time basis

Design Decisions

Expert Capacity

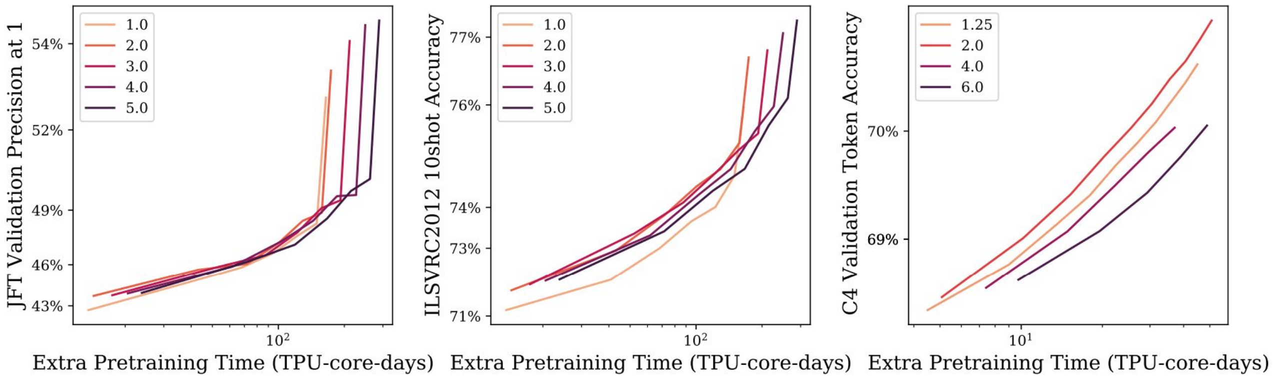
- vision experiments shown in left and center plot, language experiments shown in right plot



Design Decisions

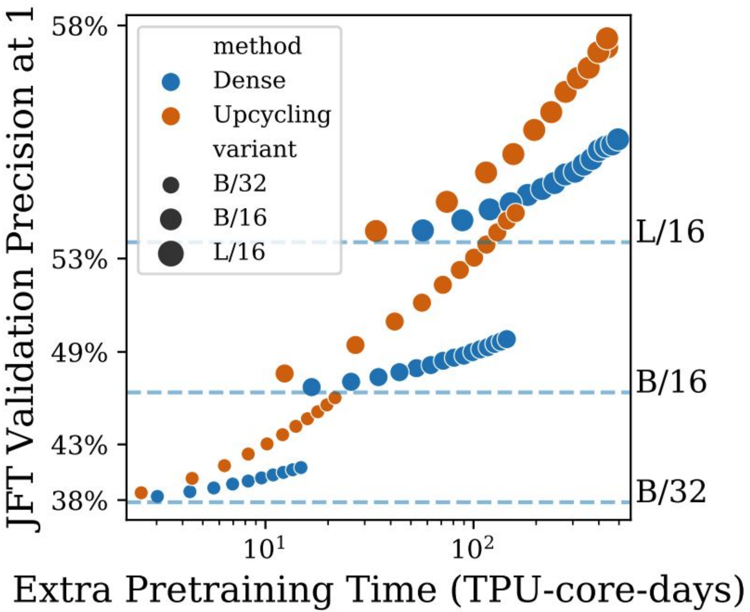
Expert Capacity

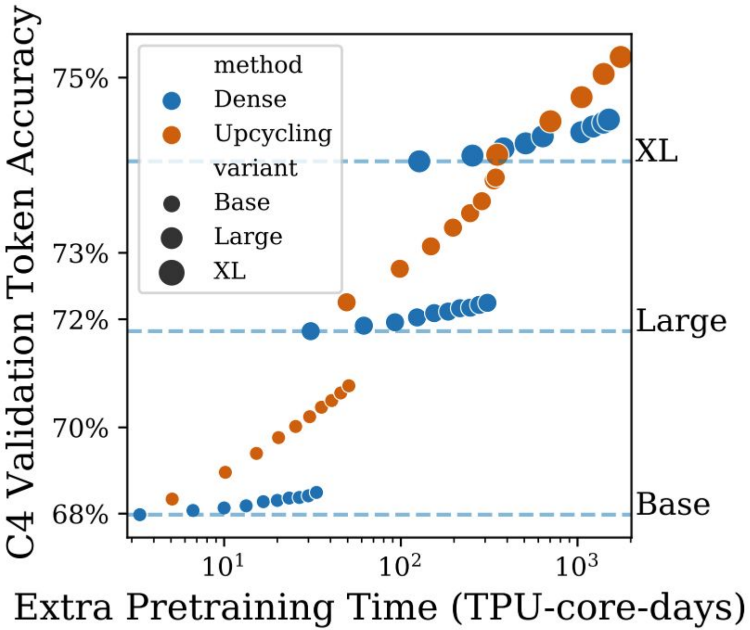
- Vision:
  - higher C returns better performance if extra computation costs are ignored
  - for lower extra pretraining time, C = 2 and C = 3 offer better performance
- Language
  - expert capacity of C = 2 is the best option regardless of allowable compute time



Experimental Results

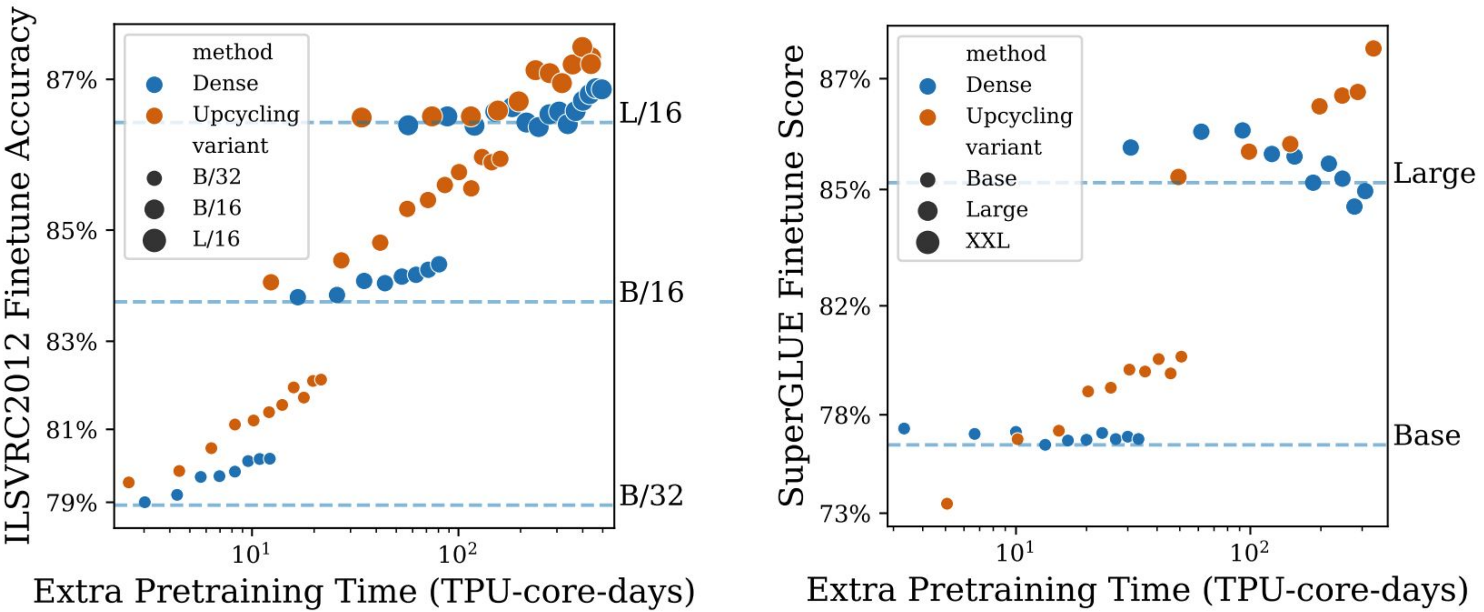
Comparison of metrics from upcycled models and dense continuation models at various sizes for vision (left) and language (right)





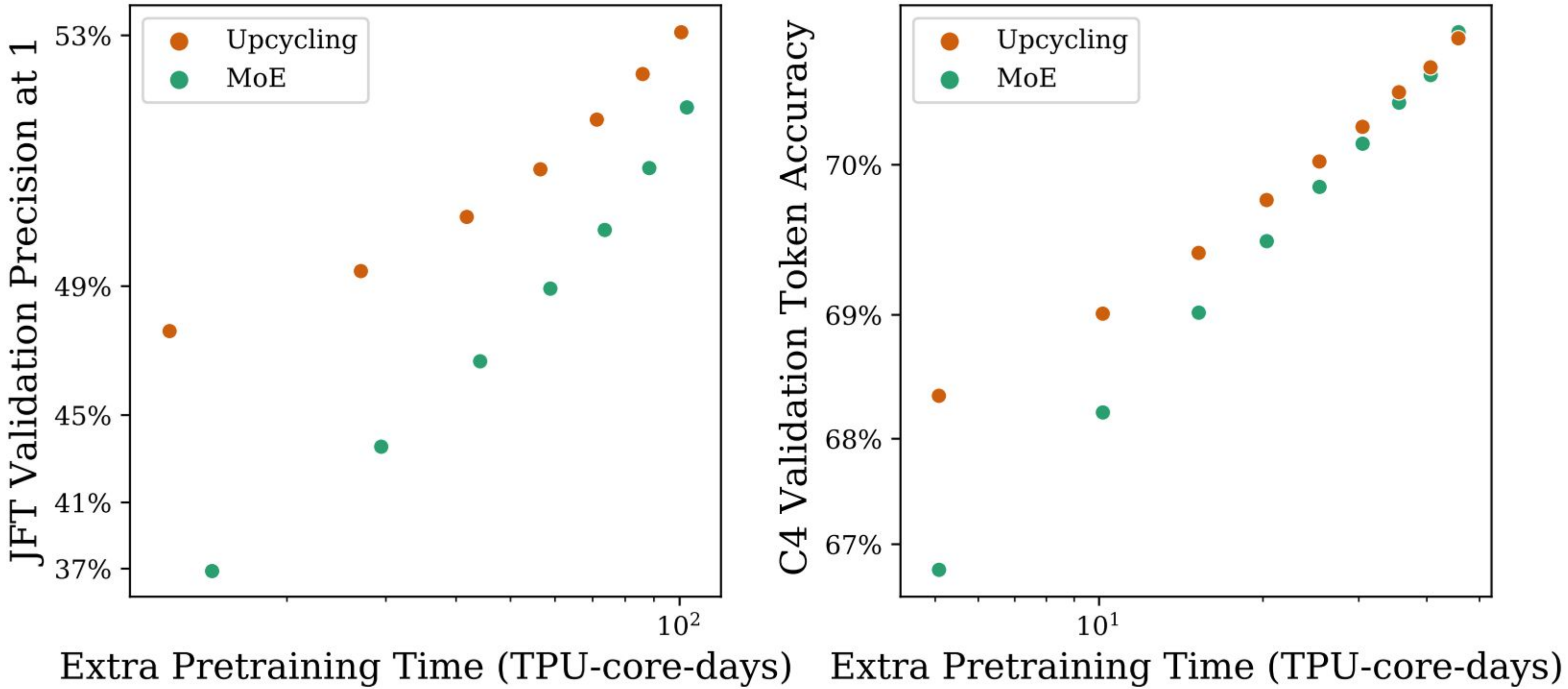
Experimental Results

Performance of dense continuation and upcycling methods after full finetuning of previous results for different size vision (left) and language (right) models



Experimental Results

Performance of sparse upcycling models vs sparse models trained from scratch



Thank you!

Q&A

References

1. LoRA: Low-Rank Adaptation of Large Language Models
2. Sparse Upcycling: Training Mixture-of-Experts from Dense Checkpoints
3. The Llama 3 Herd of Models
4. LIMA: Less Is More for Alignment

Back Up

Lima: Less is More for Alignment

Background

- LLMs learn general purpose representations of information that can be transferred into any language understanding or generation task
- this transfer is enabled by aligning language models

- primary methods include:
    - instruction tuning over multi-million example datasets
    - reinforcement learning from human feedback

Problem

- existing alignment methods require significant amounts of compute and specialized data (over multiple millions pieces of data)

- however, a strong pretrained language model can achieve strong performance by finetuning on a small set of carefully selected training examples

- alignment can be a simple process where the model learns the style of interacting with humans

Superficial Alignment Hypothesis

“A model’s knowledge and capabilities are learnt almost entirely during pre-training, while alignment teaches it which subdistribution of formats should be used when interacting with users.”

- if the hypothesis is correct, then we can sufficiently tune a pretrained language model with a small set of examples, drastically reducing training time

Methods

- collect a dataset of 1000 prompts and responses where outputs are "stylistically aligned" but the inputs are diverse
- sources come from community Q&A forums and manually created examples such as:
  - Stack Exchange, wikiHow, Pushshift Reddit Dataset

- LIMA is trained starting from LLaMa 65B and finetuned on the 1000 example alignment training set
- evaluation is done by comparing to leading language models including
  - OpenAI RLHF-based DaVinci003
  - 65B parameter reproduction of Alpaca
  - GPT 4

Methods

- training data of 1000 sequences (~750,000 tokens)

Source	#Examples	Avg Input Len.	Avg Output Len.
Training			
Stack Exchange (STEM)	200	117	523
Stack Exchange (Other)	200	119	530
wikiHow	200	12	1,811
Pushshift r/WritingPrompts	150	34	274
Natural Instructions	50	236	92
Paper Authors (Group A)	200	40	334
Dev			
Paper Authors (Group A)	50	36	N/A
Test			
Pushshift r/AskReddit	70	30	N/A
Paper Authors (Group B)	230	31	N/A

Evaluation

- Compare to 5 baselines: Alpaca 65B, LLaMa 65B, DaVinci003, Bard, Claude, GPT4
- Generate a single response from each prompt (limit 2048 tokens)
- Present prompt and 2 possible responses to human annotators
    - annotators label whether one response was better or if neither was significantly better than the other
- Apply inter-annotator agreement
    - agreements on annotations measured between:
        - crowd-crowd
        - author-author
        - crowd-author
        - crowd-GPT4
        - autho-GPT4

Results

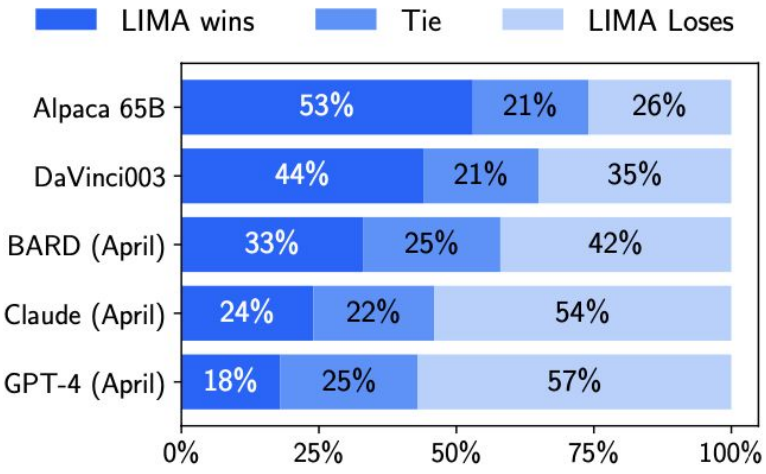


Figure 1: Human preference evaluation, comparing LIMA to 5 different baselines across 300 test prompts.

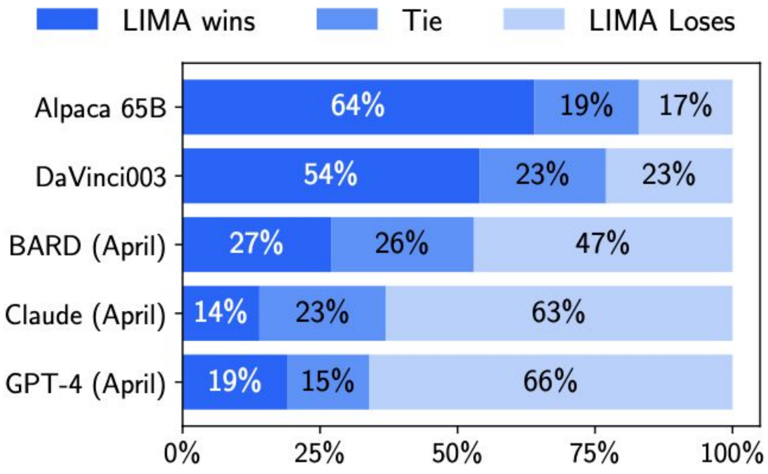


Figure 2: Preference evaluation using GPT-4 as the annotator, given the same instructions provided to humans.

Results

- Alpaca 65B tends to produce less favorable results than LIMA

- DaVinci003 produces less favorable results than LIMA despite being trained with RLHF, which is regarded as the superior alignment method

- Claude and GPT4 generally perform better than LIMA
- there are a non-trivial amount of cases where LIMA outperforms Claude and GPT4
- GPT4 prefers LIMA outputs over its own outputs 19% of the time

Limitations

- Fine-tuning on a small, carefully created set of examples can create impressive results

- However, limitations include:
    - difficult to scale the mental effort and manual labor to curate the examples
    - LIMA is not as robust as product-grade models
        - unlucky samples in decoding or adversarial prompts can lead to weak responses