



Low-rank Adaptation & RLHF

Mehar Singh, Parth Raut, Zach Eichenberger

LoRA: Low-Rank Adaptation of Large Language Models

Hu et. al. 2021

Fine Tuning is still important for LLMs

- Fine tuning Large language models on downstream tasks gives significant performance benefits
- + 20 to 50 accuracy pts.
- Want to adapt model to many different downstream tasks

Task	Dataset	GPT-3 Few-shot	GPT-3 Fine-tuned
Q&A	SQuAD V2 (F1)	69.8%	88.4%
Textual Entailment	RTE (Acc)	69%	85.4%
NL2SQL	WikiSQL (Acc)	20%	73%
	Spider (Acc)	18%	62%

[chart: Hu et al., 2021]

Why Not Just Fine Tune?

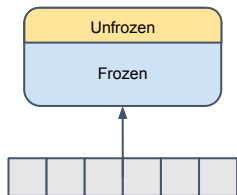
- A lot of overhead for largely similar behavior
- Storage cost: each adaptation is very large
 - $\geq 350\text{GB}$ for each adaptation
 - * many checkpoints
- Difficult to swap b/t models \rightarrow significant overhead

Parameter Efficient Model Adaptation

Fine tuning fewer weights:

E.g. [Li & Liang, 2021], [Zaken et al., 2021]

- Retrain a subset of model's weights
 - Li & Liang: train only last 3 layers
 - Zaken et al. train only biases

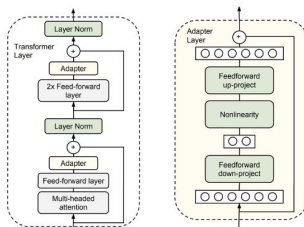


- still not very parameter efficient
- somewhat limited effectiveness

Adapter layers:

E.g. [Houlsby et al. 2019], [Lin et al. 2020], [Pfeiffer et al. 2021], [Rücklé et al., 2020]

- Add additional, smaller learned layers in between frozen model weights

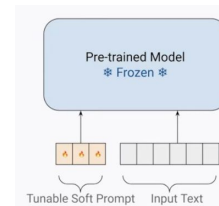


- adds latency for train & eval (20 - 30%)

Prefix tuning

E.g. [Li & Liang, 2021]

- Train soft prompt before input text
 - Either learn just token level or all hidden states



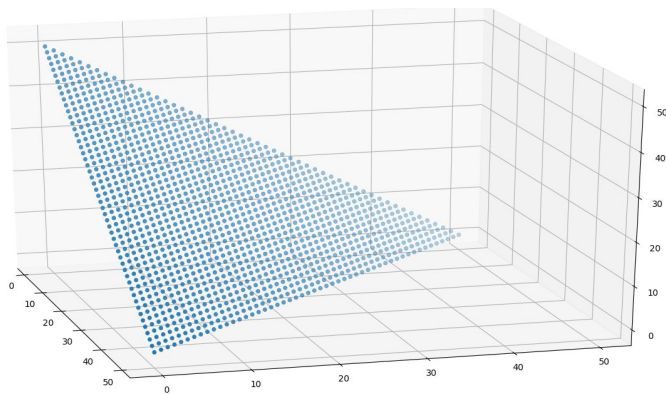
- only limited effectiveness
- limits sequence length

[diagrams 2 & 3: <https://www.cs.princeton.edu/courses/archive/fall22/cos597G/lectures/lec06.pdf>]

Review: Matrix Rank

- Rank: dimension of matrix vector space
- # of independent variables parametrized by M
- “Maximum amount of info encoded in matrix”

E.G. Rank 2 matrix in 3D:



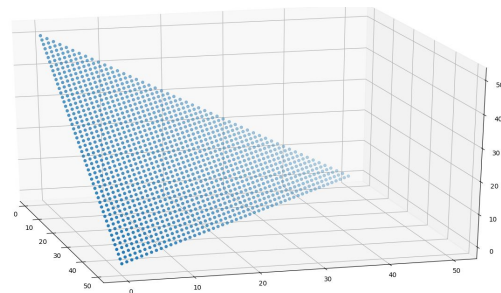
[diagram: <https://stackoverflow.com/users/8105551/matt>, 2020]

Review: Matrix Decomposition

Things to know:

for $A \in \mathbb{R}^{n \times m}$, $\text{Rank}(A) \leq \min(n, m)$

$\text{Rank}(AB) \leq \min(\text{Rank}(A), \text{Rank}(B))$



And the reverse also works:

$M = BA$ where $M \in \mathbb{R}^{m \times n}$; $A \in \mathbb{R}^{m \times r}$; $B \in \mathbb{R}^{r \times n}$

[diagram: <https://stackoverflow.com/users/8105551/matt>, 2020]

Insight: Many networks over-parametrize

- Most networks approximately learn lower dimensional weight matrices
 - Aghajanyan: “Intrinsic dimensionality” - Minimum dimensionality for success on task is much lower than weight mat’s
 - Allen-Zhu: Weight matrices of LLM’s can be re-parametrized to lower rank

A Convergence Theory for Deep Learning
via Over-Parameterization

Zeyuan Allen-Zhu
zeyuan@csail.mit.edu
Microsoft Research AI

Yuanzhi Li
yuanzhil@stanford.edu
Stanford University
Princeton University

Zhao Song
zhaos@utexas.edu
UT-Austin
University of Washington
Harvard University

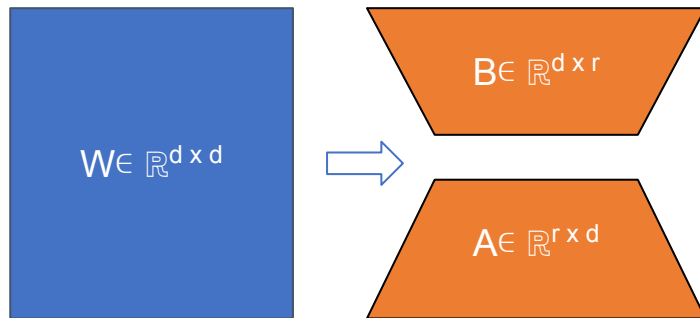
November 9, 2018
(version 5)*

INTRINSIC DIMENSIONALITY EXPLAINS THE EFFECTIVENESS OF LANGUAGE MODEL FINE-TUNING

Armen Aghajanyan, Luke Zettlemoyer, Sonal Gupta
Facebook
{armenag,lsz,sonalgupta}@fb.com

Finetune at that **Lower RAnk**?

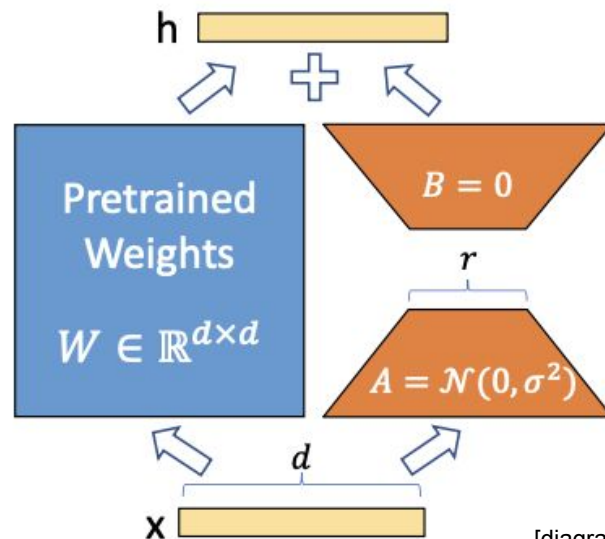
- Force training at lower rank (for speed)
- Induce a pinch to enforce lower dimension
- No. params drops from d^2 to $2dr$



Reformulation: “learn the delta”

- Instead of training whole model, train Adapter for current layer

$$\begin{aligned} h &= Wx \\ h &= W_0x + \Delta Wx \\ h &= W_0x + BAx \\ &= (W_0 + AB)x \end{aligned}$$



[diagram: Hu et al., 2021]

Reformulating the fine tuning process:

Adapt a model to a specific corpus

- Language model objective:

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

- Adapted objective: “Learn just the delta”

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

Reformulation is a generalization

Choose any $W \in \mathbb{R}^{d \times d}$ of rank r :

Then, we can write $W = AB$, where $A \in \mathbb{R}^{d \times r}$; $B \in \mathbb{R}^{r \times d}$
by rank factorization.

Experiments

- Only applied LoRA finetuning to attention layers (typically W_q and W_v)
- Kept $r = (8-64)$; smaller for small models
- Three tasks:
 - **GLUE**: standard NLP classification task suite (many tasks)
 - **Text Generation**: generate text; evaluate on metrics
 - **MultiNLI**: textual entailment task (part of GLUE)
 - **WikiSQL**: translation task for text \rightarrow sql

Performance: GLUE & Text Generation

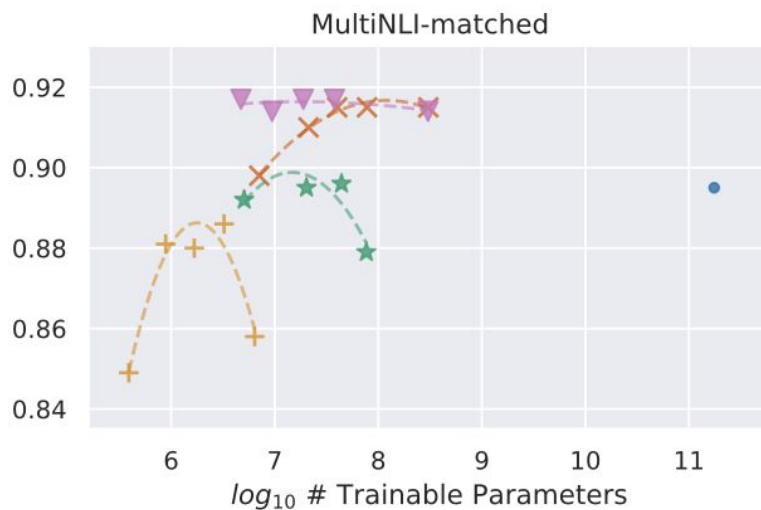
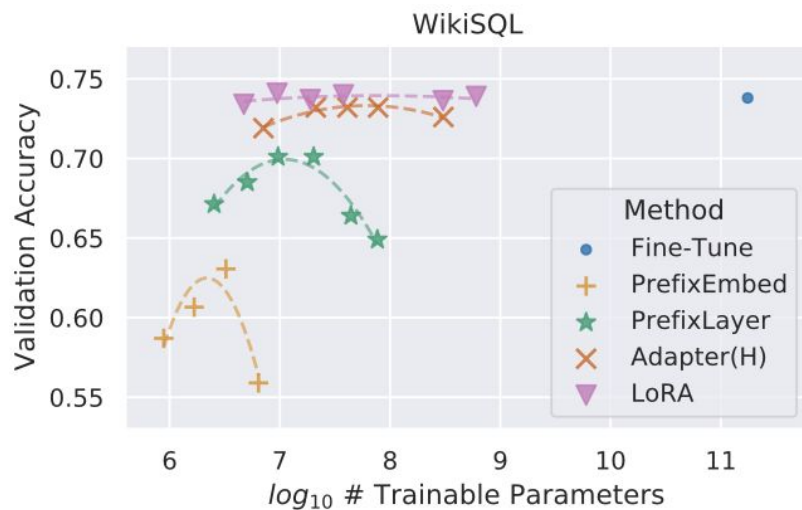
- Tested on variety of benchmarks:
 - Generally outperforms adapters, prefix-tuning methods
 - Competitive with full fine tuning

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 ^L)*	0.3M	87.1 \pm .0	94.2 \pm .1	88.5 \pm .1	60.8 \pm .4	93.1 \pm .1	90.2 \pm .0	71.5 \pm .2	89.7 \pm .3	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 \pm .2	90.3 \pm .1	85.4
RoB _{base} (LoRA)	0.3M	87.5 \pm .3	95.1\pm.2	89.7 \pm .7	63.4 \pm .2	93.3\pm.3	90.8 \pm .1	86.6\pm.7	91.5\pm.2	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\pm.2	68.2\pm.1	94.9\pm.3	91.6 \pm .1	87.4\pm.5	92.6\pm.2	89.0
RoB _{large} (Adpt ^P)†	3.0M	90.2 \pm .3	96.1 \pm .3	90.2 \pm .7	68.3\pm.1	94.8\pm.2	91.9\pm.1	83.8 \pm .2	92.1 \pm .7	88.4
RoB _{large} (Adpt ^L)†	0.8M	90.5\pm.3	96.6\pm.2	89.7 \pm .2	67.8 \pm .5	94.8\pm.3	91.7 \pm .2	80.1 \pm .2	91.9 \pm .4	87.9
RoB _{large} (Adpt ^H)†	6.0M	89.9 \pm .5	96.2 \pm .3	88.7 \pm .2	66.5 \pm .4	94.7 \pm .2	92.1 \pm .1	83.4 \pm .1	91.0 \pm .7	87.8
RoB _{large} (Adpt ^H)†	0.8M	90.3 \pm .3	96.3 \pm .5	87.7 \pm .7	66.3 \pm .0	94.7 \pm .2	91.5 \pm .1	72.9 \pm .2	91.5 \pm .5	86.4
RoB _{large} (LoRA)†	0.8M	90.6\pm.2	96.2 \pm .5	90.2\pm.1	68.2 \pm .1	94.8\pm.3	91.6 \pm .2	85.2\pm.1	92.3\pm.5	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\pm.1	96.0\pm.1	92.9\pm.1	94.9\pm.4	93.0\pm.2	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

[chart: Hu et al., 2021]

LoRA has good parameter efficiency



[graph: Hu et al., 2021]

LoRA only needs a very small r

	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

[chart: Hu et al., 2021]

Systems perspective: Learning the ΔW

- ~5000x parameter reduction over standard Finetuning
- Only need to Train, Store A,B for each W
 - Vram reduced by $\frac{2}{3}$ vs standard fine tuning (with small r)
 - Checkpoint size: 1000x reduction: (350gb -> 35MB)
- Concept: Able to store, use multiple different LoRA adaptations cheaply
- No necessary additional latency

S-LORA: Serving Thousands of Concurrent LoRA Adapters

Sheng et. al. 2023

S-LoRA at a High Level

- **Goal:** How to scalably serve thousands of LoRA adapters on a single machine
- Efficient Batched Inference with many LoRA adapters
 - **1. Heterogeneous Batching:** custom CUDA kernels for non-contiguous memory
 - **2. Memory Management:** reducing memory fragmentation and increasing batch size
 - **3. Tensor Parallelism:** a novel tensor parallelism strategy for effective parallelization across multiple GPUs

1. Heterogeneous Batching and Scheduling

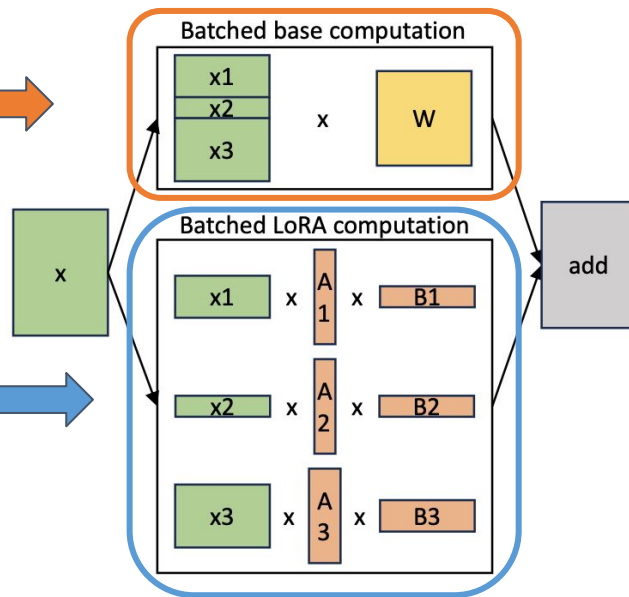
$$h = xW' = x(W + AB)$$

$$= xW + xAB.$$

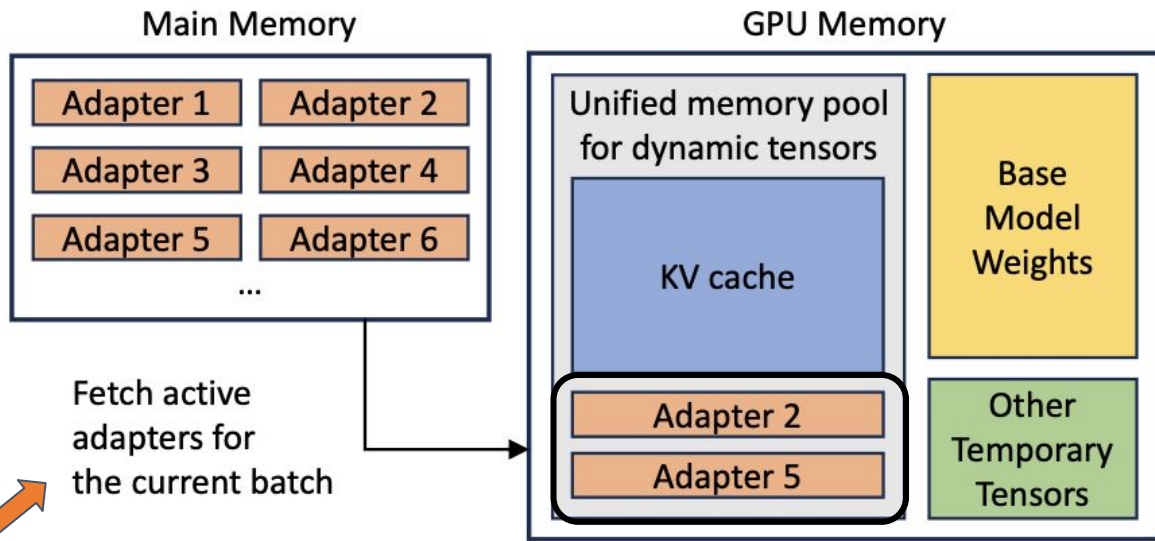
Compute this in parallel

Batching implemented by GEMM

Required custom CUDA kernels due to various sequence lengths and adapter lengths



1. Heterogeneous Batching and Scheduling



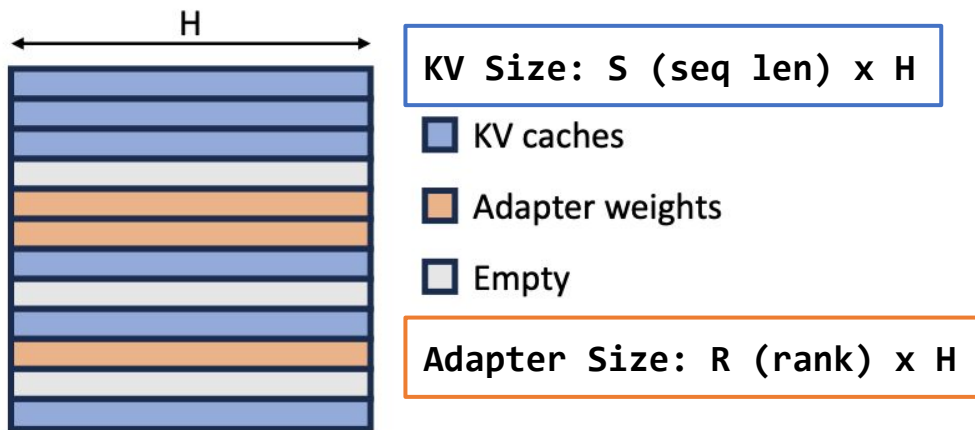
Batching: Store LoRA Adapters in main memory and fetch on demand

Adapter Clustering: prioritize batching requests that use the same adapter

Admission Control: Abort strategy when traffic is higher than serving system capacity. Estimate set of latest requests that can be served and respond in order of arrival time, or drop request.

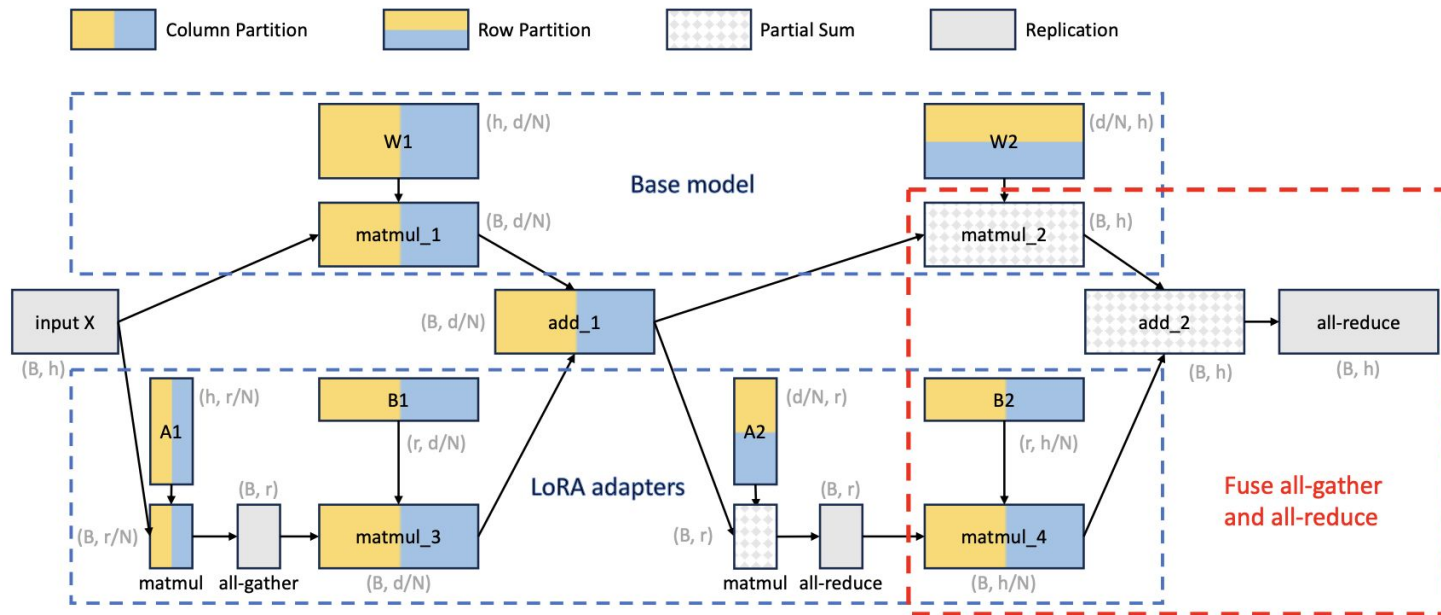
2. Memory Management

- **Memory Fragmentation** from dynamic loading and off loading of adapter weights and **Latency Overhead** from adapter loading and off loading



- **Unified Paging:** KV Cache management (PagedAttention) very similar to LoRA Adapters, manage both
- **Prefetching and Overlapping**
 - Look at current waiting queue and prefetch needed adapters

3. Tensor Parallelism



per-GPU
shape in
gray
B: number of
tokens
h: input dim
N: number of
devices
d: hidden
size
r: adapter
rank

3. Tensor Parallelism - Communication Cost

$$\frac{2(N-1)Bh}{N}$$

Base model (one all-reduce)

B: number of tokens
h: input dim
N: number of devices
d: hidden size
r: adapter rank

Added LoRA
(three all-gather for query, key, value projections, and one all-reduce for output projection)

$$3 \frac{(N-1)Br}{N} + \frac{2(N-1)Br}{N} = \frac{5(N-1)Br}{N}$$

Additional communication cost introduced by LoRA is negligible because:

$$r \ll h$$

Results - Comparing Benchmarks

Throughput (req/s)

Model Setup	n	S-LoRA	vLLM-packed	PEFT
S1	5	8.05	2.04	0.88
	100	7.99	OOM	0.25
	1000	7.64	OOM	-
	2000	7.61	OOM	-
S2	5	7.48	2.04	0.74
	100	7.29	OOM	0.24
	1000	6.69	OOM	-
	2000	6.71	OOM	-
S4	2	4.49	3.83	0.54
	100	4.28	OOM	0.13
	1000	3.96	OOM	-

PEFT: HuggingFace PEFT, lacks advanced batching and memory management

vLLM-packed: Simple multimodal serving solution, but does not support LoRA. LoRA weights were merged with base model.

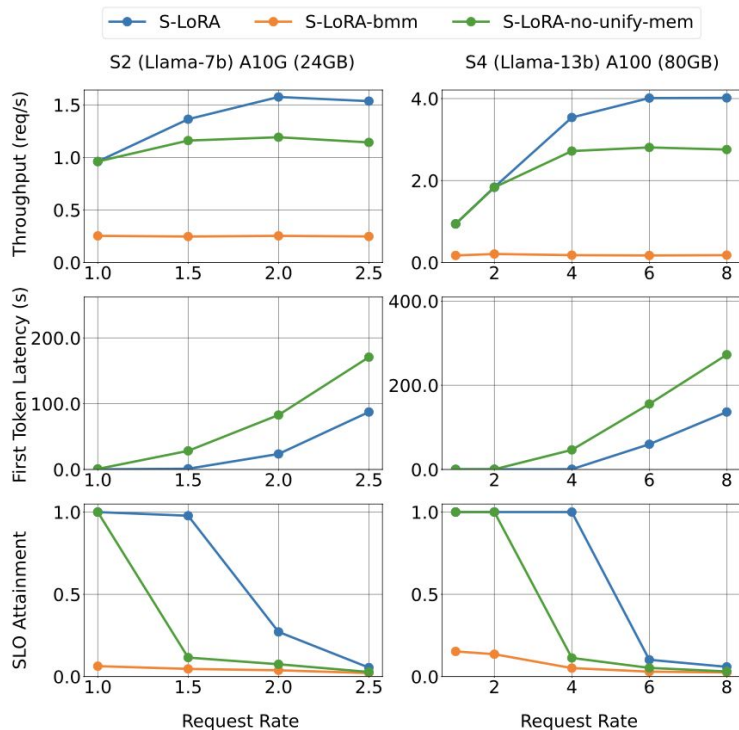
OOM: Out-of-memory :(

Results - Synthetic Workload*

Full S-LoRA

S-LoRA without Unified Paging
and customized kernels

S-LoRA without Unified Paging



*Generated using the Gamma process.
Covered various combinations of
variables to simulate real workloads.
Values used shown in table below:

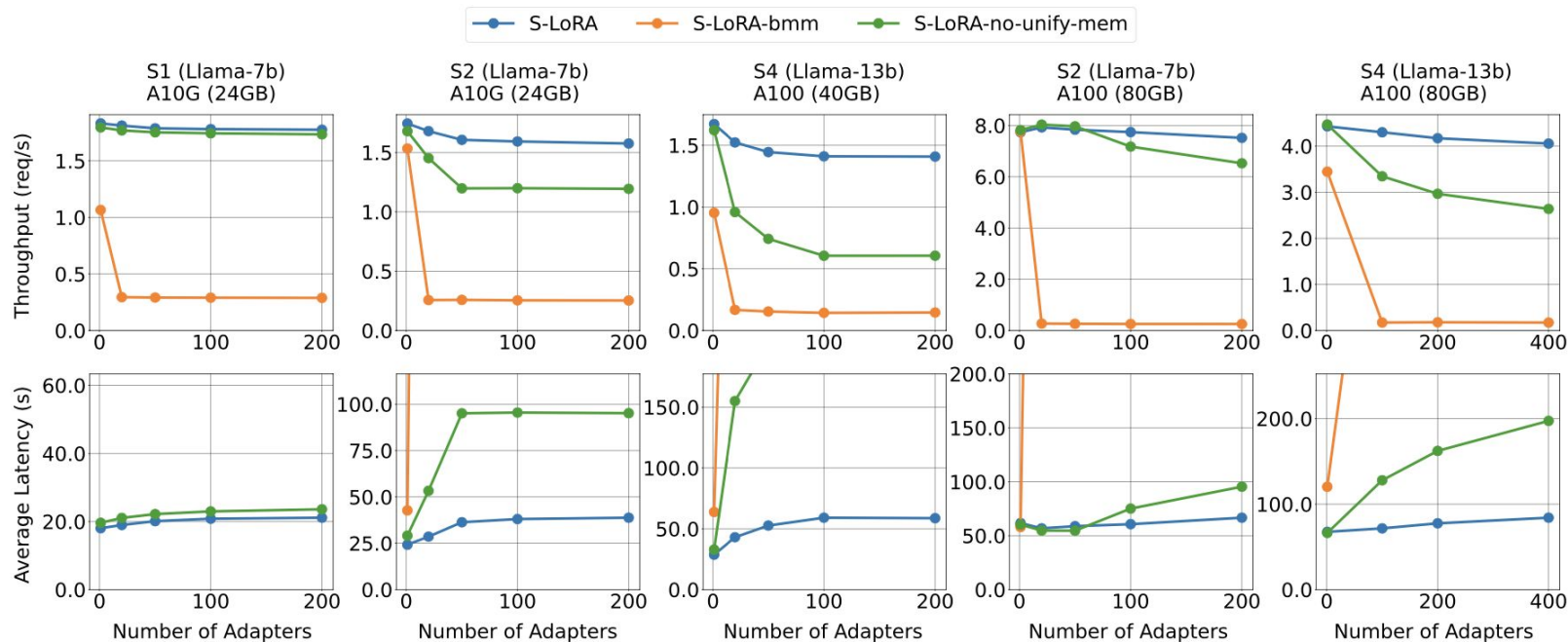
Setting	n	α	R	cv	$[I_l, I_u]$	$[O_l, O_u]$
7B @ A10G (24G)	200	1	2	1	[8, 512]	[8, 512]
7B @ A100 (80G)	200	1	10	1	[8, 512]	[8, 512]
13B @ A100 (40G)	200	1	2	1	[8, 512]	[8, 512]
13B @ A100 (80G)	400	1	6	1	[8, 512]	[8, 512]

Results - Synthetic Workload

Full S-LoRA

S-LoRA without Unified Paging
and customized kernels

S-LoRA without Unified Paging

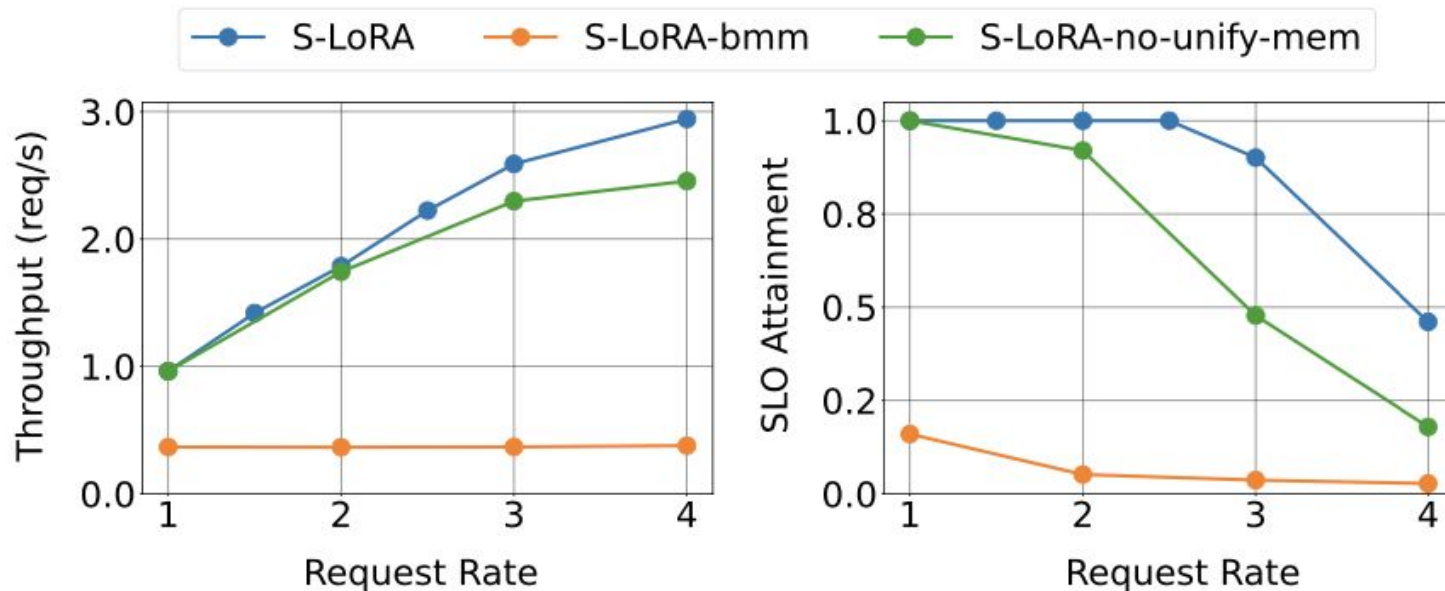


Results - Real Workload

Full S-LoRA

S-LoRA without Unified Paging
and customized kernels

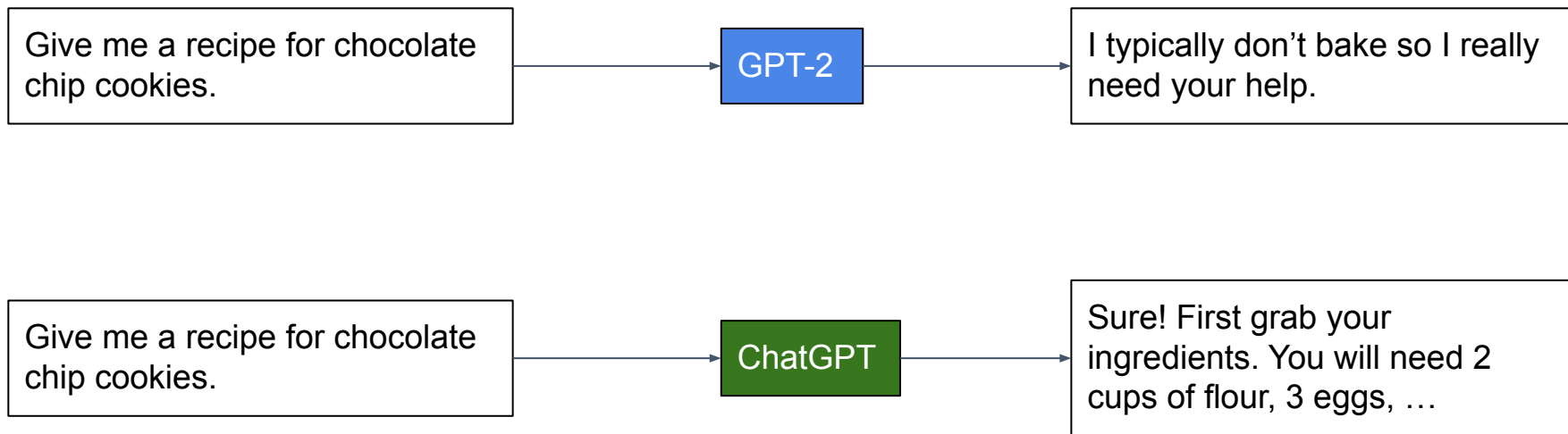
S-LoRA without Unified Paging



Training language models to follow instructions with human feedback

Ouyang et. al. 2022

Traditional LMs \neq Useful LMs



Previous Methods for Alignment

Previous methods for alignment are hyper-specific to downstream task:

Supervised fine-tuning (SFT)

- Text summarization (Ziegler et al., 2019; Stiennon et al., 2020; Böhm et al., 2019; Wu et al., 2021)

Human feedback with Reinforcement Learning (RL)

- Dialogue (Jaques et al., 2019; Yi et al., 2019; Hancock et al., 2019)
- Translation (Kreutzer et al., 2018; Bahdanau et al., 2016)
- Semantic parsing (Lawrence and Riezler, 2018)
- Story generation (Zhou and Xu, 2020)
- Review generation (Cho et al., 2018)
- Evidence extraction (Perez et al., 2019).

SFT Cons

- 1) Single “ground-truth” for each input prompt. In reality, many outputs may be considered “correct.”

- 2) Loss for next-token prediction is at a token level
 - a) Input prompt: “Chocolate chip cookies are ”
 - b) Consider 2 possible LM predictions: “sugary” & “spicy”
 - c) Ground truth: “sweet”
 - d) **error(sugary) = error(spicy)**

Steps for Human Instruction Following

- 1) Do **supervised fine-tuning (SFT)** with human labeler ground-truth
 - a) But, unlike previous works, fine-tune on a wide range of tasks
 - b) Also known as instruction-tuning
- 2) Collect comparison data & train a **reward model (RM)**
- 3) Optimize LM against reward model (**RLHF**)

OpenAI named their resulting model **InstructGPT**.

Step 1: Supervised Fine-tuning

1) Collect M input prompts to LM

- a) $\mathbf{X} = \{x_i\}$ for $i = 1$ to M
- b) [Early InstructGPT] OpenAI used labelers to generate prompts
- c) [Newer InstructGPT] OpenAI used prompts submitted to OpenAI API

2) For each input prompt, collect labeler-provided demonstrations of desired behavior on input prompts

- d) $\mathbf{Y} = \{y_i\}$ for $i = 1$ to M

Step 1: Supervised Fine-tuning

3) Create supervised fine-tuning (SFT) dataset

- a) $\mathbf{D} = \{(x_i, y_i)\}$ for $i = 1$ to M
- b) Each sample is a pair: (input prompt, desired output)

4) Fine-tune with next token prediction!

Task Distribution for SFT 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%

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: ""

Step 2: Train Reward Model (RM)

Why do we need a reward model?

Next-token prediction training is insufficient! (as seen through the cons of SFT)

- It doesn't directly capture semantic meaning

A reward model (if trained well) can output a reward on a LM's output that semantically captures whether their output as a whole is good or bad.

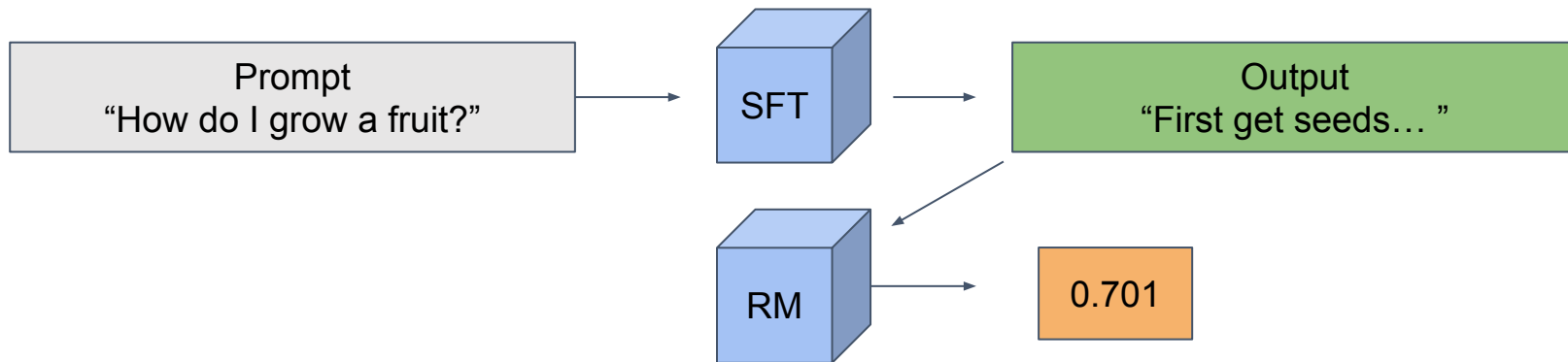
- Not at a token level

Step 2: Train Reward Model (RM)

Intuition:

Instead of a single desired response for each prompt input & optimize to minimize loss...

What if we can have a reward score for each LM output & optimize to maximize reward



Reward Model Dataset Collection

Collect dataset (2 options)

- 1) Sample SFT outputs and a human assigned reward score
 - a) Using labelers for text-to-reward is noisy (people have different preferences/scales)

- 2) **Better:** Collect comparison data from SFT output
 - a) Dataset = $\{(x, y_{\text{winner}}, y_{\text{loser}})\}$
 - b) $y_{\text{winner}}, y_{\text{loser}}$ are LM outputs for the same input prompt x

Train Reward Model (RM)

$$\text{loss}(\theta) = -\frac{1}{\binom{K}{2}} E_{(x, y_w, y_l) \sim D} [\log(\underbrace{\sigma(r_\theta(x, y_w) - r_\theta(x, y_l))}_{\text{High if predicted reward for winner} > \text{predicted reward for loser}})]$$

↓
Negation because
loss is a measure
of how bad a
model is

High if predicted reward for winner >
predicted reward for loser

In other words, high if reward model is
good

Step 3: RL with Human Feedback (RLHF)

We have:

- 1) Supervised fine-tuned model (SFT)
- 2) Reward model (RM)

Let's initialize a Policy Model as a copy of our SFT model. Policy model will be updated using RLHF

How to optimize Policy model to maximize reward?

- Use a class of RL algorithms called Policy Gradient methods
 - PPO is one such alg. that OpenAI used

RLHF Training Objective

Want a high reward

$$\text{objective}(\phi) = \overbrace{E_{(x,y) \sim D_{\pi_{\phi}^{\text{RL}}}} [r_{\theta}(x,y)]} + \underbrace{\gamma E_{x \sim D_{\text{pretrain}}} [\log(\pi_{\phi}^{\text{RL}}(x))]}$$

Want to maintain next-token
prediction capabilities (traditional
training objective)

Preventing Hallucinations

What are hallucinations in the context of RLHF?

- When our policy models learn to fool the reward model

How to prevent reward hacking?

- Don't overfit to reward model (causes hallucinations)
- After RLHF, we want resulting Policy model to not be too far away from the SFT model it was initialized as

Enforcing “closeness” to SFT model

- Keep a copy of the SFT model (pre-trained and frozen)
- When updating the Policy model, check that it's not far off from the SFT model

KL Divergence Term

How do we compute similarity of Policy and SFT model?

- KL Divergence
 - A measure of alignment between output token distributions of Policy and SFT model
 - High KL divergence = low similarity
 - Low KL divergence = high similarity

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

Full RLHF Training Objective

Want a high reward

Want a low KL Divergence wrt. SFT model

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

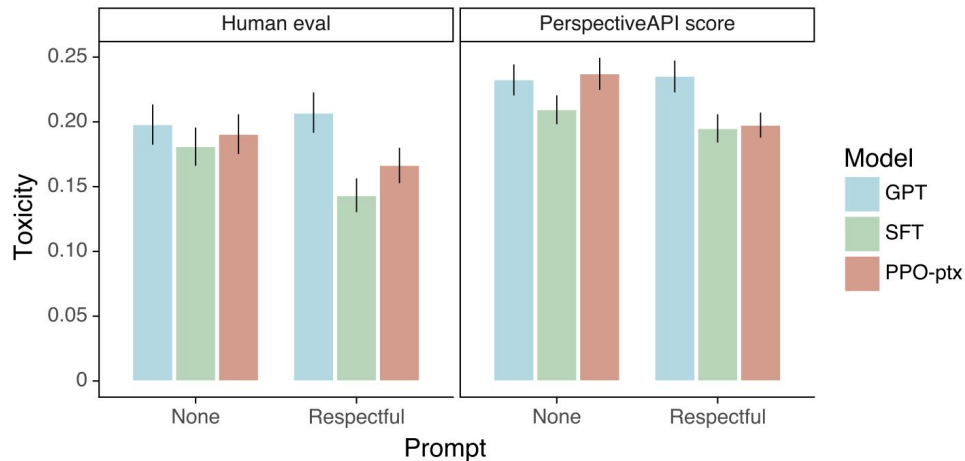
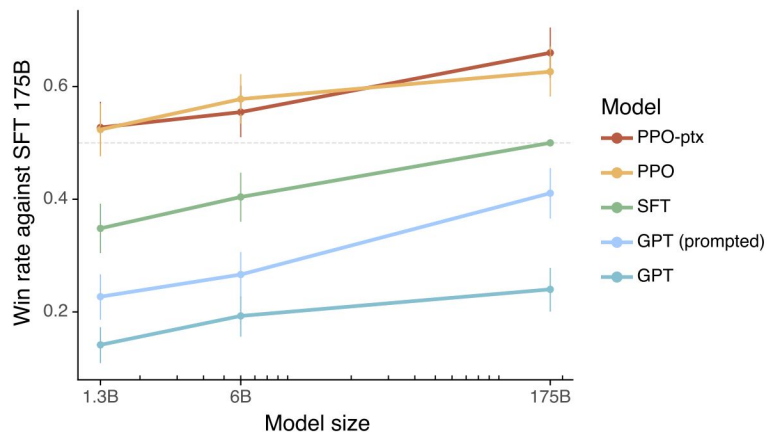
Want to maintain next-token prediction capabilities (traditional training objective)

Results: Win Rate & Toxicity

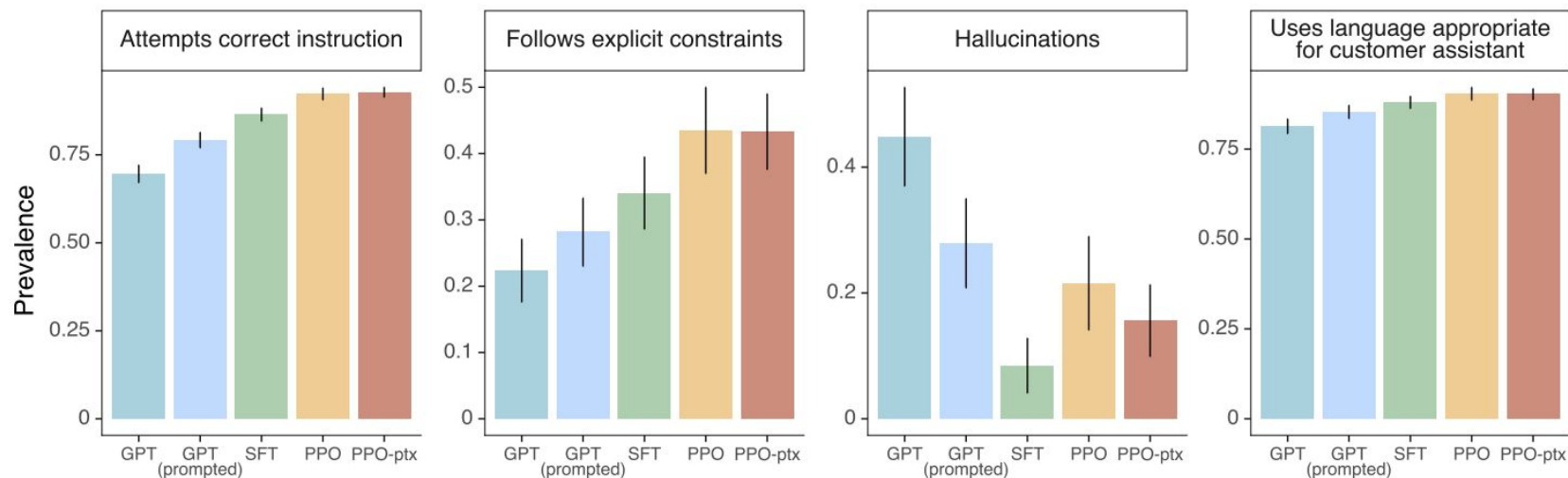
PPO-ptx (red) is their RLHF model

SFT (green) is their supervised fine-tuned model

GPT (blue) is their baseline model



Results: Usefulness & Hallucinations


















Results Summarized

- 1) Labelers significantly refer InstructGPT over GPT (85% of the time)
- 2) InstructGPT is more reliable and easier to control
- 3) InstructGPT models show improvements in truthfulness & toxicity over GPT-3
- 4) InstructGPT hallucinates less than GPT-3 (a 21% vs. 41% hallucination rate, respectively).

Systems Impact: Inference

ChatGPT vs. GPT-3

- 20b vs. 175b parameters
- ChatGPT better at conversational tasks and faster

GPT MODELS COMPARISON CHART					
Model	Size	Memory capacity	Accuracy	Input formats	Price
GPT-3	 175B	 1,500 words	 <60%	 Text, speech	
GPT-3.5	 20B	 8,000 words	 <60%	 Text, speech	
GPT-4 <small>greenice</small>	 >1T (?)	 25,000- 64,000 words	 >80%	 Text, speech, image	

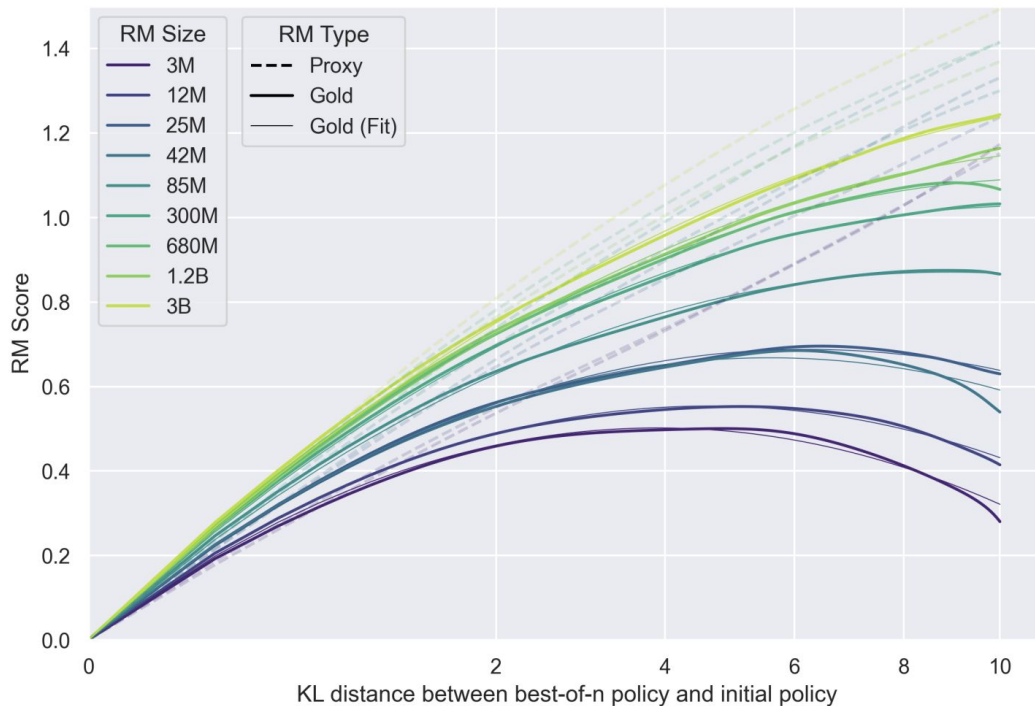
Is Bigger Better? Why The ChatGPT Vs. GPT-3 Vs. GPT-4 'Battle' Is Just A Family Chat", Farzeev 2023
Figure: <https://greenice.net/chatgpt-vs-gpt-4-vs-gpt-3/>

Systems Impact: Training

- RLHF stage requires:
 - forward pass outputs for SFT, Policy Model, RM
 - gradients from backwards pass
 - optimizer weights (ex: AdamW)
- Aligning LLaMA-2 7B eight Nvidia A100 80GB GPUs
- A single node with eight A100 GPUs costs ~\$200K using AlpacaFarm
- AlpacaFarm library uses fully sharded data-parallel training to distribute parameters across GPUs
 - It also uses oracle LLMs to provide human feedback and cut costs for human labelers

“Exploring the impact of low-rank adaptation on the performance, efficiency, and regularization of RLHF”, Sun et. al. 2023

Even Reward Models can be Massive



"Scaling Laws for Reward Model Overoptimization", Gao et. al. 2022

LoRA for RLHF

Reduce memory consumption with LoRA on LLaMA-2 7B:

- Rank: 8
- Layers applied: Query, Key, Values, and Output weights
- Params trained: 16.7M parameters with LoRA
 - 0.2% of LLaMA 7B's parameters

Results:

- Higher preference win rate for LoRA than full-PPO (47.5% vs. 46.7%)
- Only required 10 hours of training on two A100s

"Exploring the impact of low-rank adaptation on the performance, efficiency, and regularization of RLHF", Sun et. al. 2023

LoRA on Regularization for RLHF

Recall from RLHF: RLHF requires KL regularization between Policy & SFT model

- make sure Policy model is close to SFT model
- don't "overfit" to Reward model

Recall from LoRA: LoRA only trains residual (delta) weights.

- Original weights from pretraining are frozen.

"Exploring the impact of low-rank adaptation on the performance, efficiency, and regularization of RLHF", Sun et. al. 2023

LoRA on Regularization for RLHF

Intuition: Since LoRA only trains delta weights, the resulting fine-tuned model will naturally be close to the pre-trained model

Key finding: When using LoRA for RLHF, the KL regularization penalty does not affect win rate!

- LoRA does heavy lifting in making sure Policy model remains close to SFT model.
- No need for forward pass through SFT model during RLHF.

Instead of needing RM, SFT, and Policy model for RLHF step...

Now we just need RM and Policy model!

"Exploring the impact of low-rank adaptation on the performance, efficiency, and regularization of RLHF", Sun et. al. 2023

Q&A