dLoRA
&
Mixture of LoRA Experts

Lohit Kamatham, Alex Ji, Vatsal Joshi, Omkar Yadav

dLoRA: Dynamically Orchestrating Requests and Adapters for LoRA LLM Serving

Authors: Bingyang Wu, Ruidong Zhu, Zili Zhang, Peng Sun, Xuanzhe Liu, Xin Jin

Affiliations: Peking University, Shanghai Artificial Intelligence Laboratory

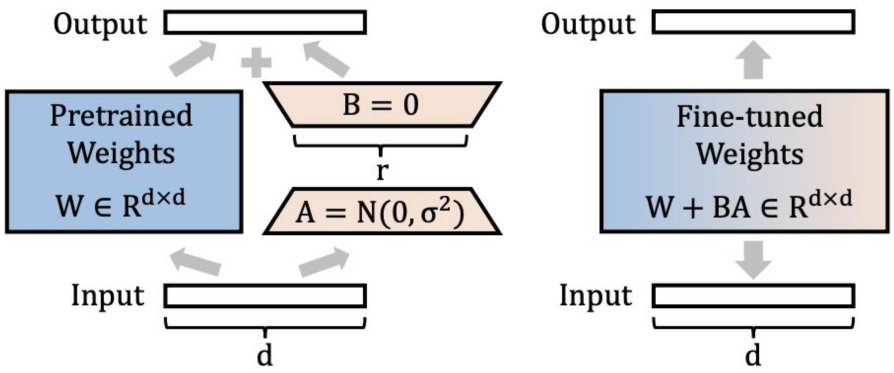
Brief Background: LoRA

LoRA: Low-rank adaptation

- A popular approach to fine-tune LLMs
- $h = Wx + BAx$

Benefits

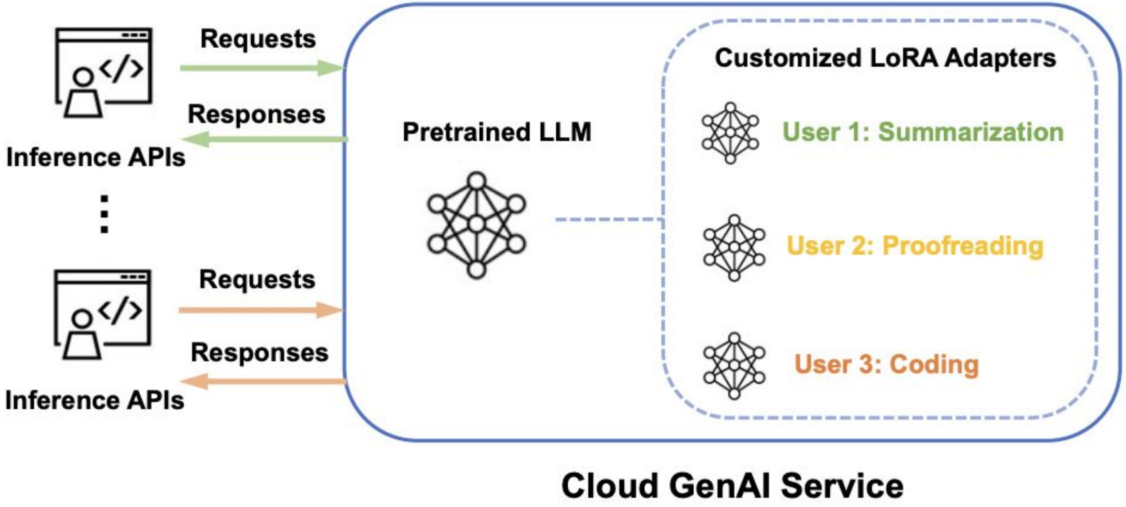
- Reduces fine-tuning costs by updating only a small portion of model parameters



(a) LoRA fine-tuning. (b) LoRA inference.

Motivation: How to Serve Different Requests

- Different users may use different adapters for different scenarios



The Problem: Issues with Current Serving Systems

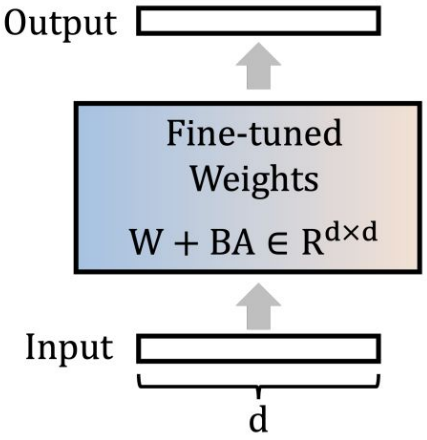
1. LLM Serving Systems:
    a. Orca, vLLM, Hugging Face PEFT
    b. **Issue:** only focuses on single LLM serving scenario
2. Traditional DNN Serving Systems:
    a. SHEPHERD, AlpaServe, PetS
    b. **Issue:** does not target autoregressive LLMs & LoRA

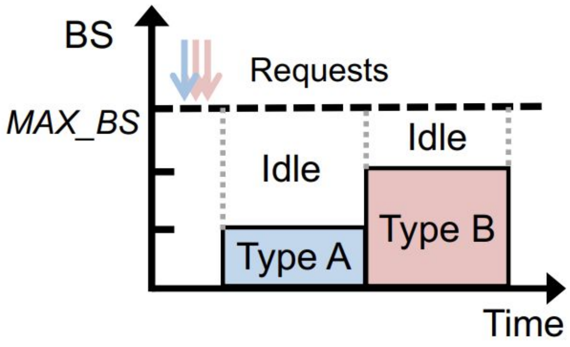
"Simply combining these solutions to serve multi-type scenarios leads to inefficiencies at the **replica** and **cluster** level"

Problem 1: Replica Level

Issue: merged inference (ex. PEFT)

Consequence: accommodates one adapter at a time -> low GPU utilization

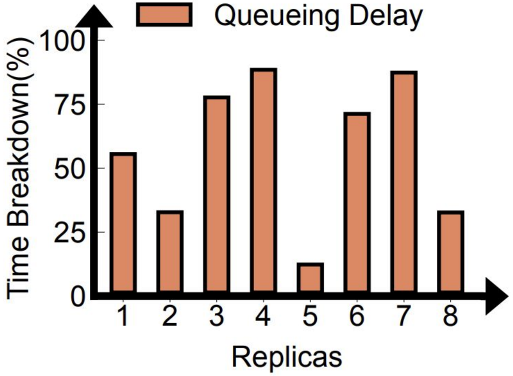




Problem 2: Cluster Level

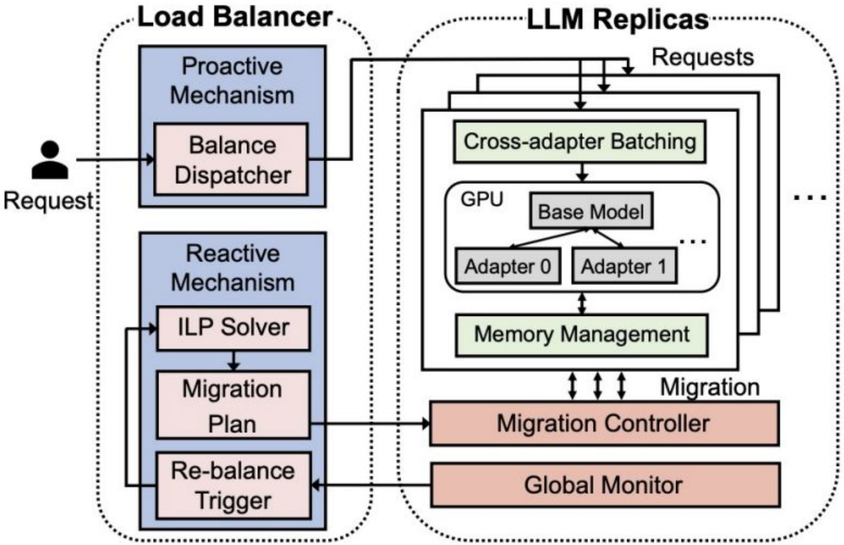
Issue: varying input and output lengths of incoming requests

Consequence: severe load imbalance between replicas due to difference in computational requirements



Solution: dLoRA

- **Overall Observation**: dynamically orchestrate LoRA adapters and requests
- **Intra-replica (replica level)**: dynamic batching + memory management
- **Inter-replica (cluster level)**: proactive dispatching + reactive migration



Solution: Intra-replica (Dynamic Batching)

Unmerged Inference: One set of pretrained weights with multiple LoRA adapters connected to it

Idea: Share the same common computation among different requests

Problem: Extra computational overhead

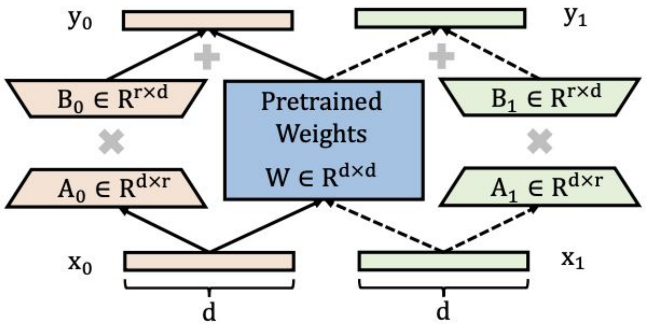
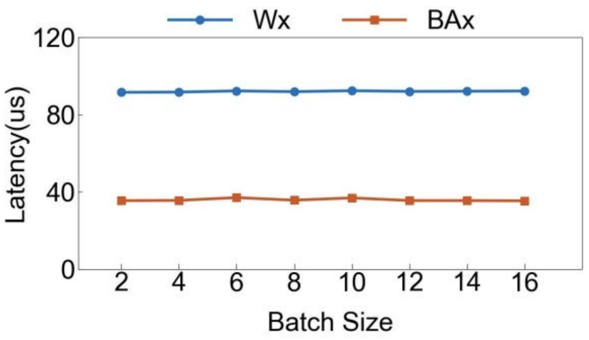


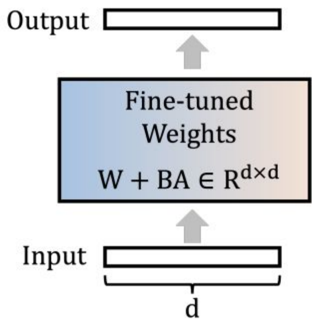
Figure 4: Unmerged inference.



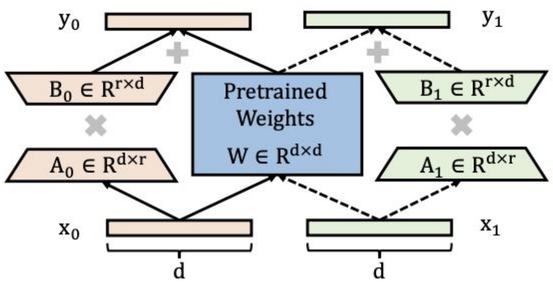
Solution: Intra-replica (Dynamic Batching)

Observation: There are tradeoffs between using merged and unmerged LoRA adapters

Idea: Use a combination of merged and unmerged LoRA adapters to balance queueing delay (merged) and computational overhead (unmerged).



No additional inference overhead, but longer queuing delay



Shorter queuing delay, but extra computational cost

Solution: Intra-replica (Dynamic Batching)

Challenge: switch overhead & scheduling overhead

Solution: Dynamic Batching Algorithm with adaptive threshold tuning and credit-based scheduling

- Amortized switching overhead across multiple future iterations (fixed cost)

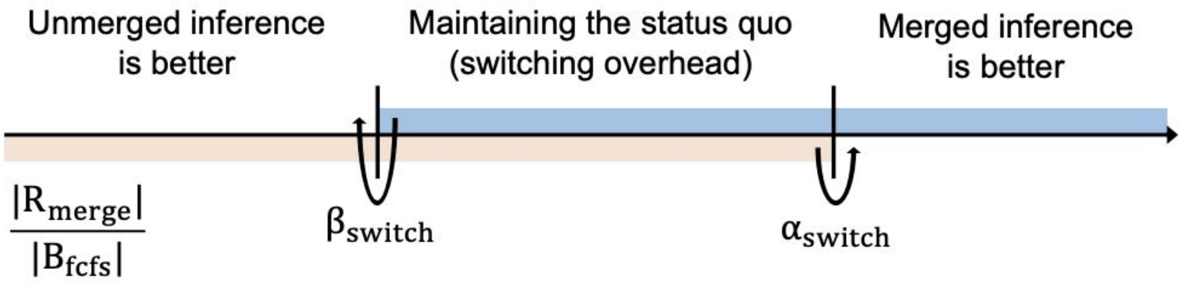


Figure 6: Thresholds demonstration.

Solution: Intra-replica (Memory Management)

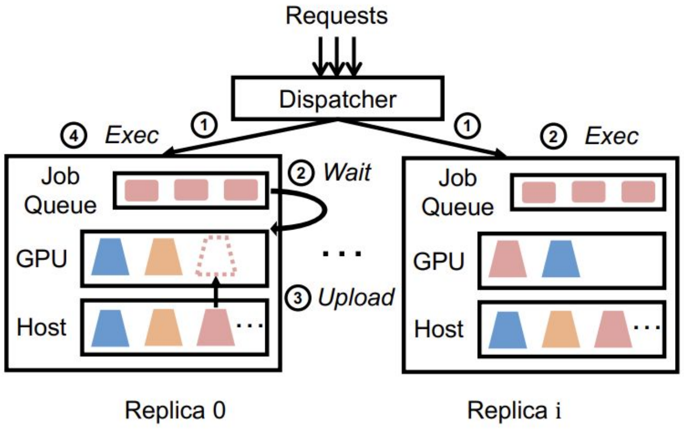
- dLoRA dynamically allocates GPU memory between adapters and requests, adjusting based on real-time usage to optimize performance.

- Unused adapters and requests are swapped to host memory, reducing GPU memory contention.

Solution: Inter-replica (Proactive Load Balancing)

Proactive Load Balancing: dLoRA uses the adapter loading time and queuing delay to balance the loads among the different LoRA replicas

Challenge: The proactive mechanism alone is not sufficient since the resource usage (i.e., input and output length) of a request is variable.

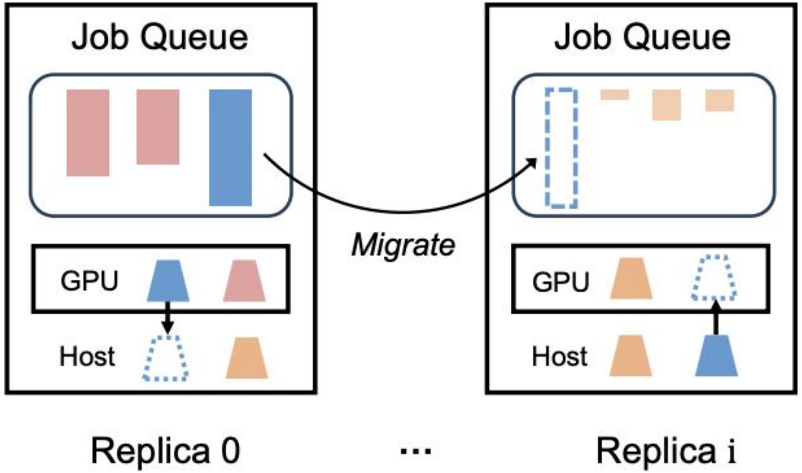


Solution: Inter-replica (Dynamic Load Balancing)

Reactive Load Balancing:

Global monitor to detect heavily loaded replicas, triggering a re-balance mechanism to optimize resource usage.

A co-migration algorithm migrates LoRA adapters and requests (with intermediate states) from overloaded replicas to others.



Evaluation: Experiment Setup

Hardware: 4 GPU cluster * 8 NVIDIA A800 80GB GPUs

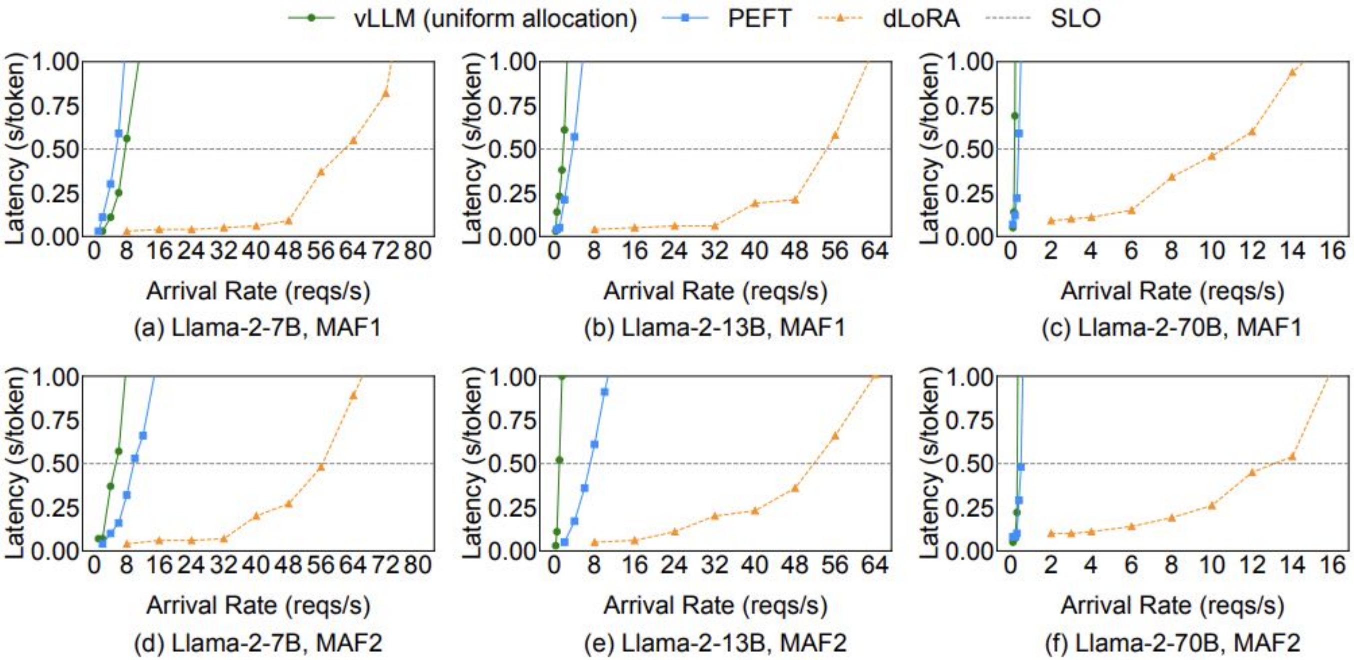
Model: Llama-2 model series as **base** model

Workload: Based on ShareGPT dataset & Azure Production (2019 & 2021) traces

Metrics: Average Latency = sum(each request's end-to-end latency)/# of tokens

Baselines: vLLM, PEFT

Evaluation: End-to-End Performance



Related Works:

LLM Serving Systems:

- FlexGen: Optimize batching, memory management, and throughput for single LLMs.
- S-LoRA, Punica: Batch requests for multiple LoRA LLMs, but lack dLoRA's dynamic load balancing.

Traditional DNN Serving Systems:

- TensorFlow Serving, Triton: General-purpose DNN serving, no LLM-specific optimizations.

Load Balancing

- Pegasus, Scarlett: Use replication/migration but don’t address dLoRA’s request-adapter dependencies.

Potential Limitations & Improvements

- Dependency on Predefined Workload Patterns
  - **Limitation:** not able to fully capture variability in real-world LLM

- Average Latency as the Primary Metric
  - **Limitation:** obscures variability in latency across requests

MoLE: Mixture of LoRA Experts

Authors: Xun Wu, Shaohan Huang, Furu Wei

Affiliations: Microsoft Research Asia, Tsinghua University

Motivation: Composing multiple LoRAs

- LoRA finetunes huge models efficiently by injecting trainable rank-decomposition matrices
- **Benefits?** Reduces computational overhead while maintaining model performance
- **Problem:** In practical applications, one LoRA falls short of meeting user expectations
- **Challenge:** How do we combine multiple trained LoRAs for joint generation, while preserving individual characteristics?

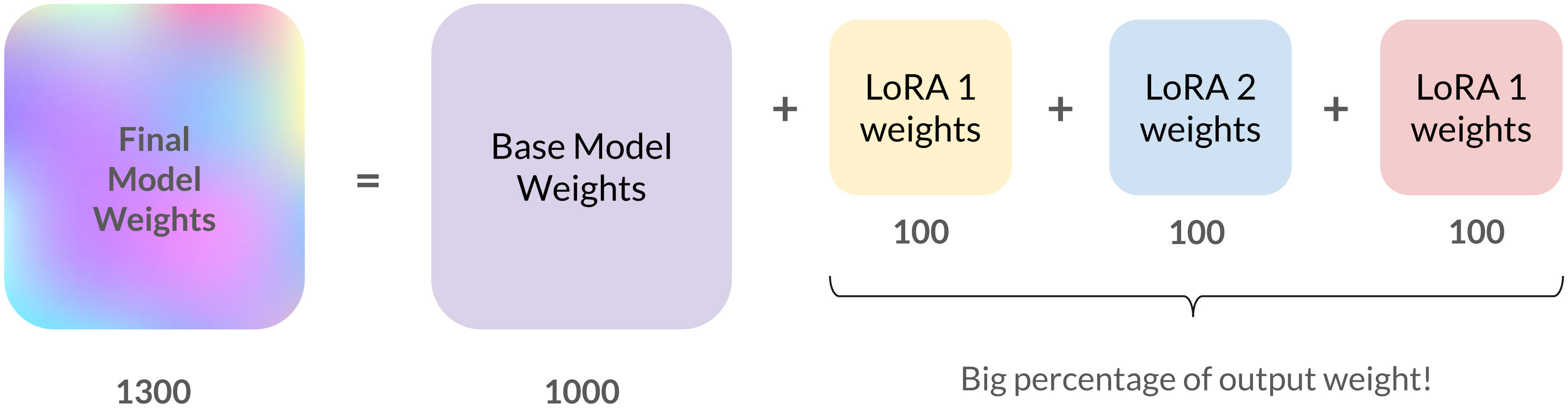
Existing Method: Linear Composition

Intuition: To gain specialized knowledge from all LoRAs, *simply add all of their weights together*

Final Model Weights = Base Model Weights + LoRA 1 weights + LoRA 2 weights + LoRA 3 weights

Existing Method: Linear Composition

Intuition: To gain specialized knowledge from all LoRAs, *simply add all of their weights together*



Existing Method: Linear Composition

Intuition: To gain specialized knowledge from all LoRAs, *simply add all of their weights together*

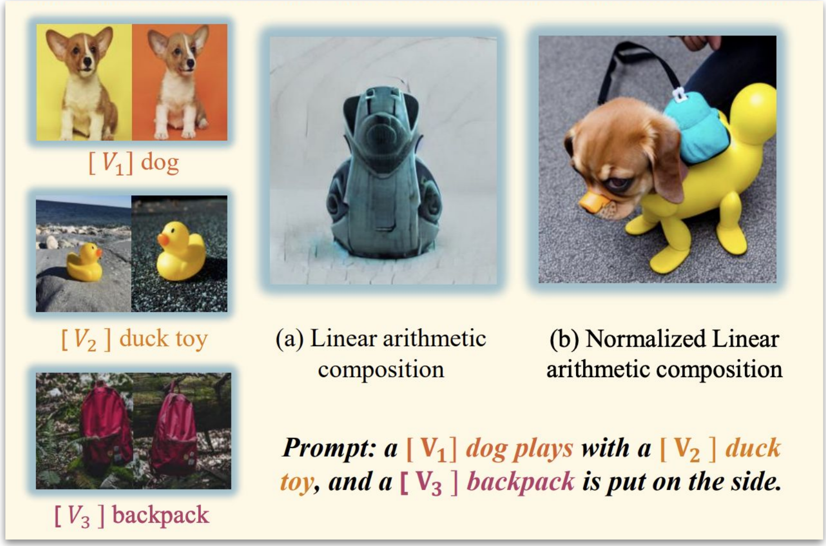
Final Model Weights = Base Model Weights + LoRA 1 weights + LoRA 2 weights + LoRA 1 weights

1300 1000 33 33 33

Normalize

Issues: Linear Composition

- Interference between LoRA modules
- Loss of LoRA characteristics
- Can't capture complex, non-linear interactions between tasks, every LoRA always contributes equally

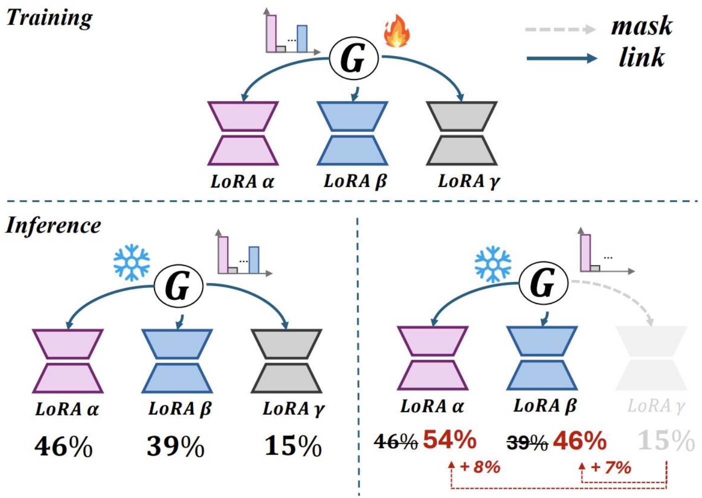


Introduction: Mixture of LoRA Experts (MoLE)

- Let's respect each LoRA as a distinct expert

- Then, let's have a **learnable gating function**

- This function will distribute weights to each LoRA to mask out undesired LoRAs



Benefits

Let's say we have 3 LoRAs pretrained on dogs, cats, and barns

Single Concept Generation: "generate an image of a dog"

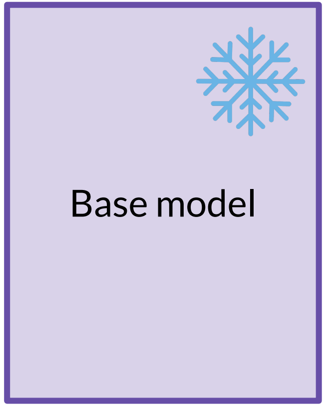
- Gating functions will prioritize the dog LoRA and masks others
- Learn contributions **dynamically** from input
- **Task-specific**
- **No dilution** of target LoRAs

Benefits

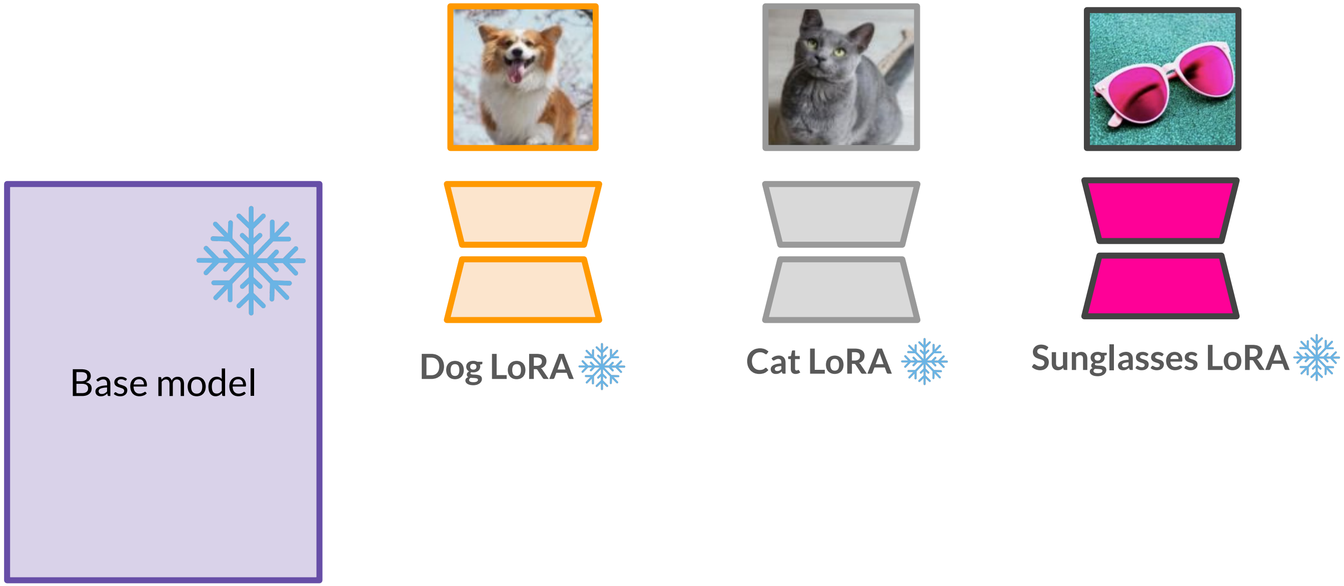
Multi Concept Generation: “generate an image of a dog next to a cat in a barn”

- Gating occurs **layer-wise**
- Gating function detects parts of the input strongly associated with "dog" features, and gives Dog LoRA a **more substantial influence** over those aspects **w/o overwhelming the contributions of the cat and barn LoRAs**
- **Complex, non-linear** relationships keep individual LoRA characteristics
- Better finetuning of concept-mixing = **better prompt alignment**

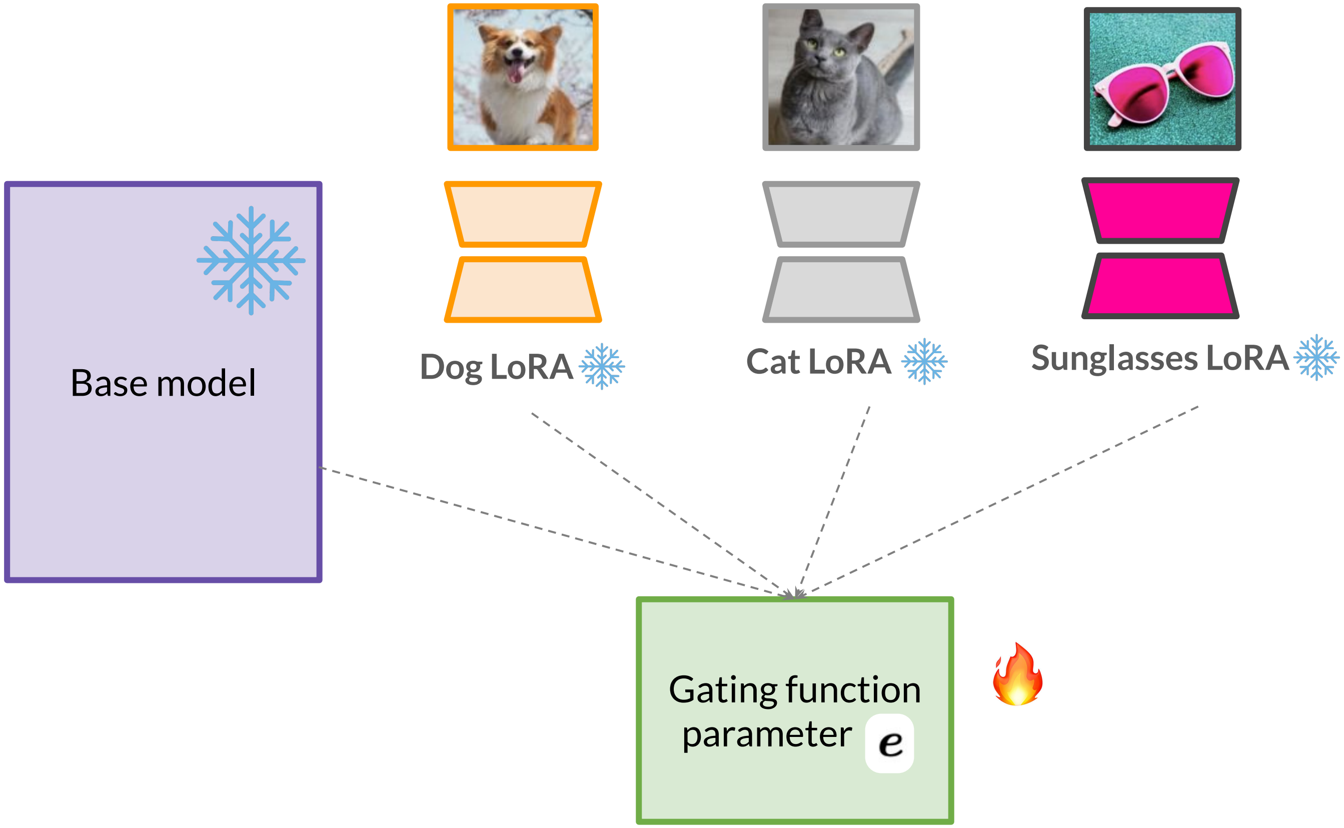
Training



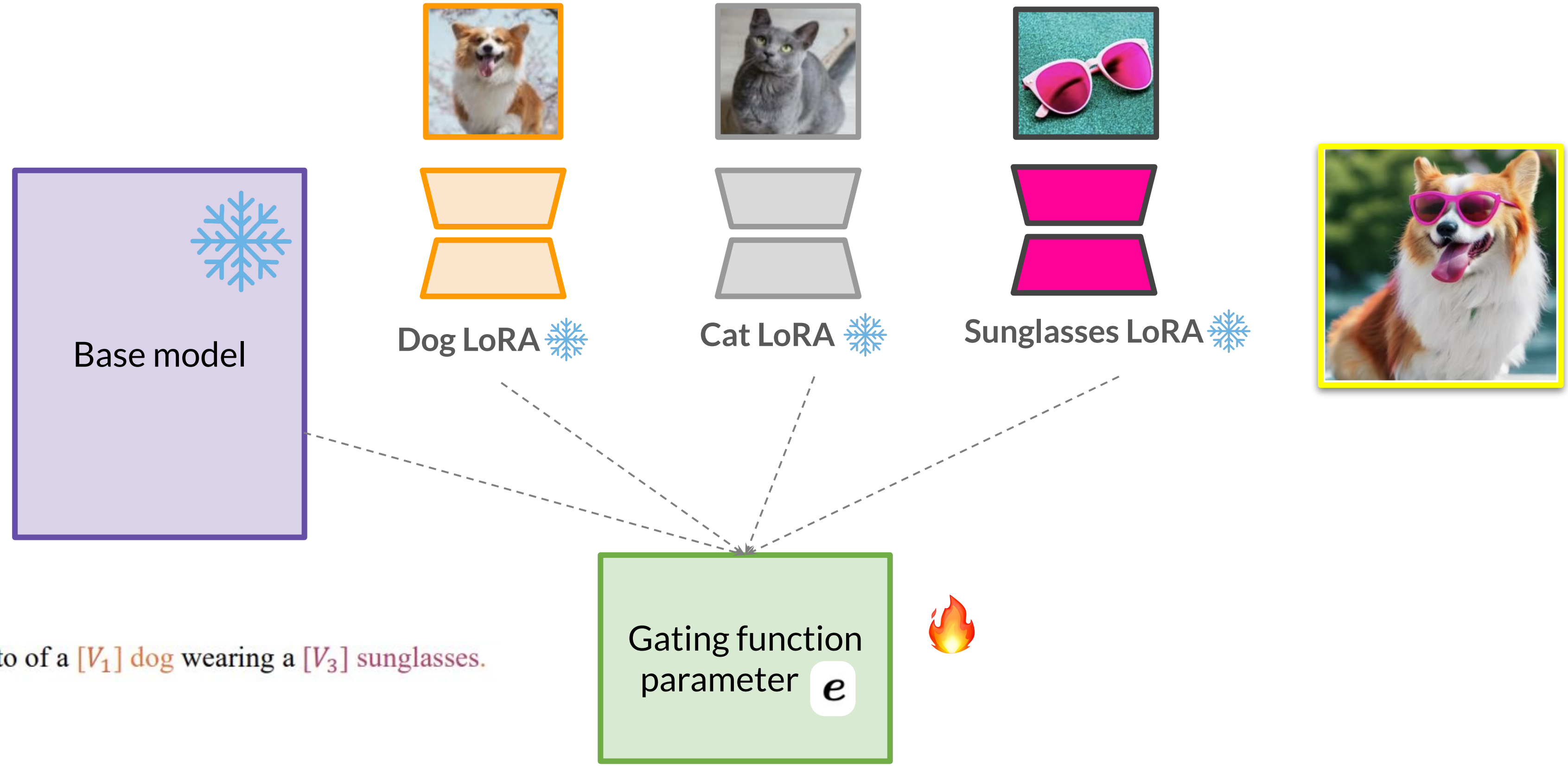
Training



Training

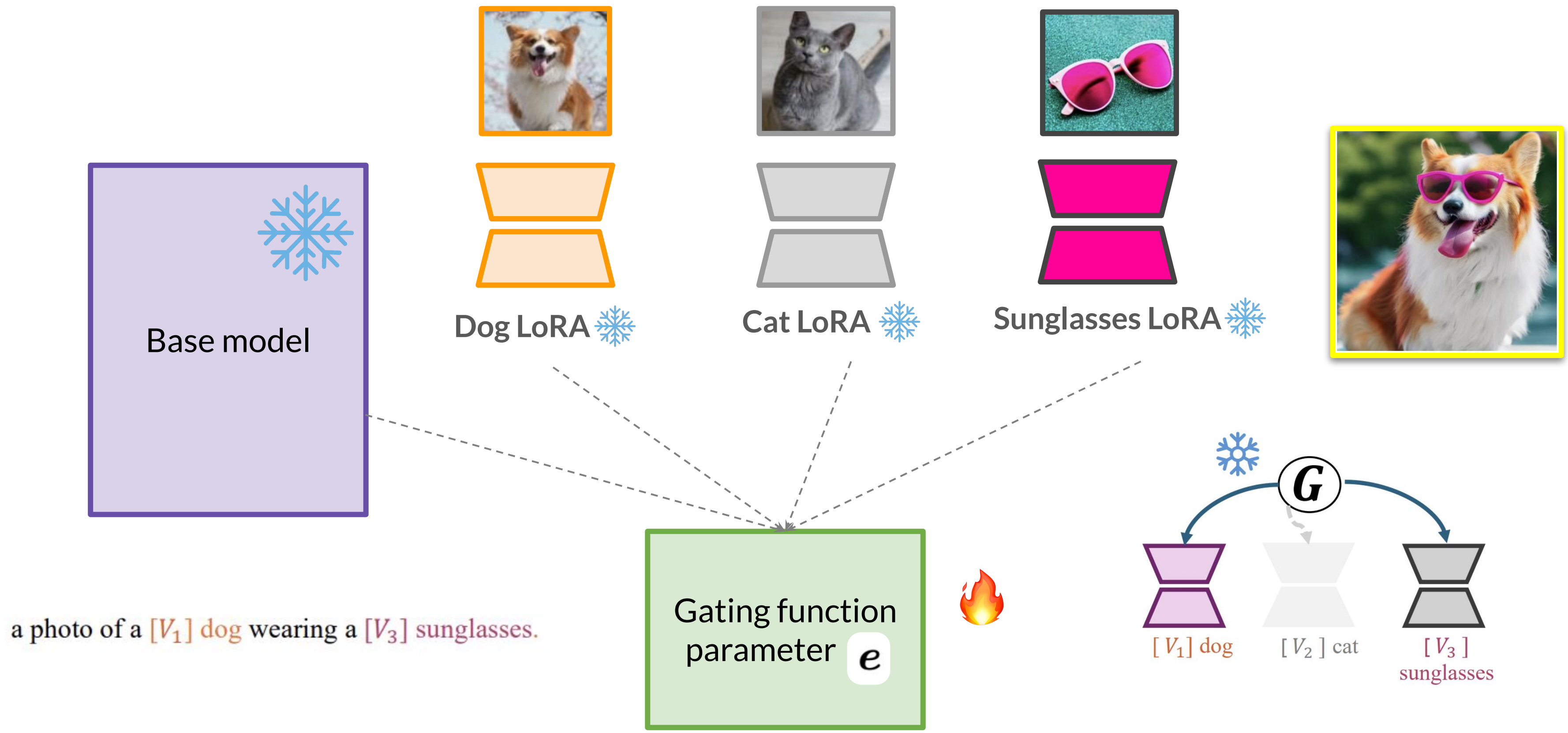

Base model
Dog LoRA
Cat LoRA
Sunglasses LoRA
Gating function
parameter e

Training

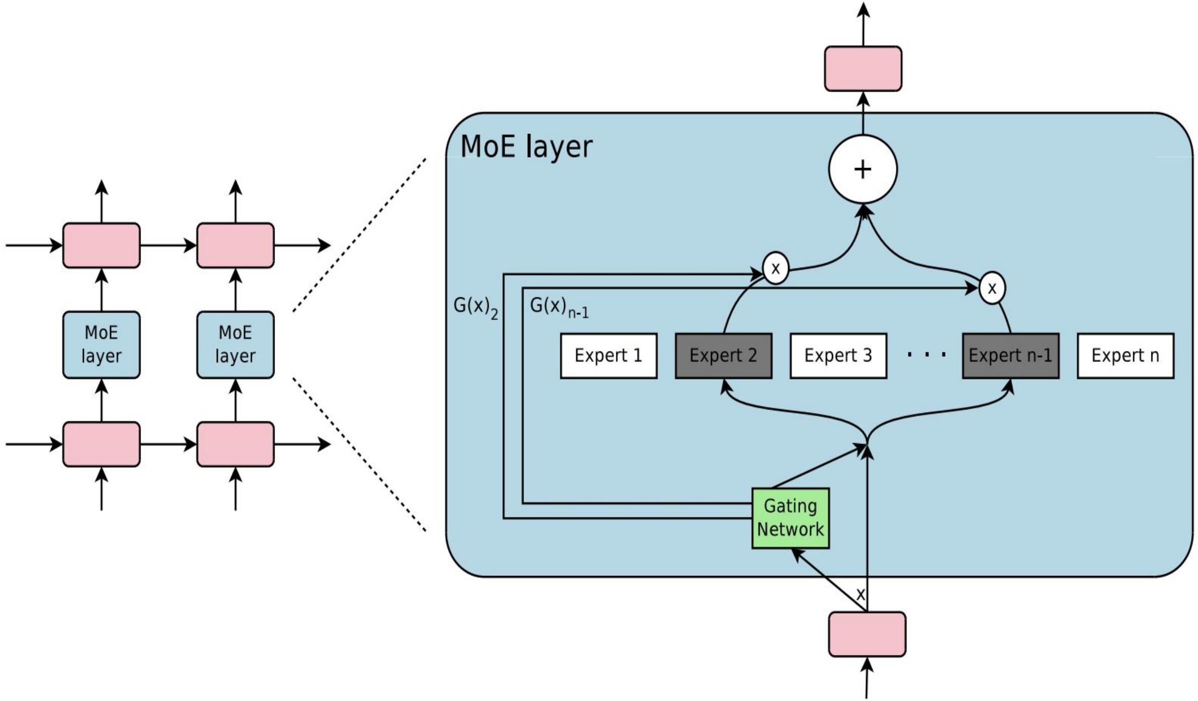


a photo of a $[V_1]$ dog wearing a $[V_3]$ sunglasses.

Training


Base model
Dog LoRA
Cat LoRA
Sunglasses LoRA
Gating function parameter e
G
[V_1] dog
[V_2] cat
[V_3] sunglasses
a photo of a [V_1] dog wearing a [V_3] sunglasses.

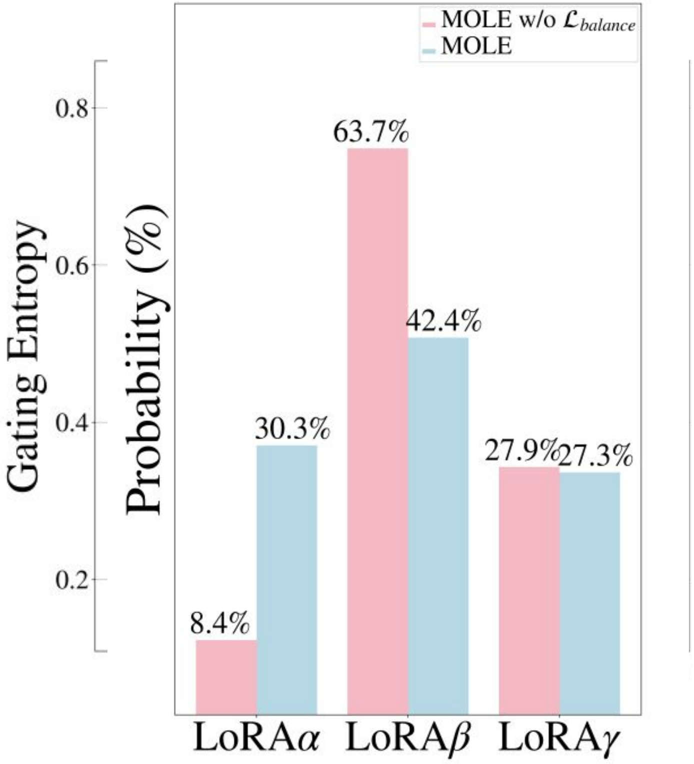
Method: Gating Balance Loss



Method: Gating Balance Loss

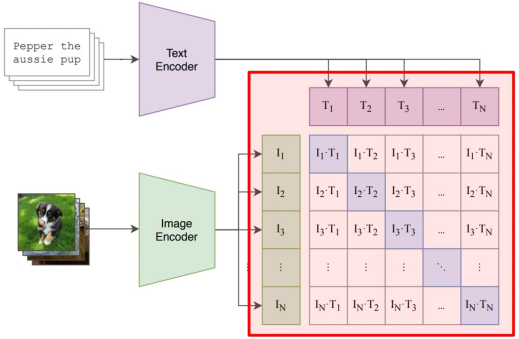
- Adding a loss term to encourage balance amongst LORAs

$$\mathcal{L}_{\text{balance}} = -\log\left(\prod_{i=0}^{N} \mathbf{q}^{(i)}\right)$$



Method: Domain Specific Loss

- Domain specific loss is used to finetune the architecture to the task that we are trying to solve
- In general these depend on the task at hand but the paper provides 2 examples:
  - V&L: Contrastive loss for both images and text
  - NLP: Cross entropy loss



$$H = -\sum p(x) \log p(x)$$

Experiments: V&L Domain

- Combining three separately trained Low-Rank Adaptations (LoRAs) for given concepts

- Uses multi concept images to incorporate multiple different components to a single image
  - I.E. a photo of a **dog** and a **cat**, with a **barn** standing nearby

- Evaluated how well the image and text generation was in comparison to other state of the art models

Results: V&L Domain

# Visual Concepts	Text-alignment			Image-alignment, (Concept 1)			Image-alignment, (Concept 2)			Image-alignment, (Concept 3)		
	NLA	SVDiff	MoLE	NLA	SVDiff	MoLE	NLA	SVDiff	MoLE	NLA	SVDiff	MoLE
Fancy boot + Monster + Clock	0.754	0.742	0.832	0.781	0.758	0.784	0.791	0.749	0.801	0.763	0.812	0.809
Emoji + Car + Cartoon	0.610	0.607	0.696	0.619	0.734	0.839	0.711	0.702	0.709	0.652	0.686	0.679
Vase + Wolf plushie + Teapot	0.752	0.812	0.863	0.687	0.807	0.835	0.705	0.782	0.746	0.653	0.694	0.721
White Cat + Wolf plushie + Can	0.704	0.772	0.780	0.801	0.804	0.802	0.678	0.763	0.825	0.650	0.729	0.714
Shiny sneaker + Wolf plushie + Teapot	0.778	0.789	0.791	0.812	0.783	0.690	0.723	0.751	0.790	0.688	0.676	0.721
Car + Wolf plushie + Teapot	0.635	0.681	0.684	0.652	0.763	0.713	0.601	0.664	0.745	0.685	0.612	0.707
Can + Wolf plushie + backpack	0.601	0.782	0.754	0.653	0.705	0.767	0.602	0.755	0.782	0.681	0.738	0.723
Golden Retriever + Wolf plushie + Teapot	0.670	0.716	0.784	0.713	0.784	0.790	0.601	0.802	0.809	0.678	0.761	0.748
Golden Retriever + Boot + Monster	0.614	0.762	0.755	0.665	0.662	0.620	0.748	0.832	0.862	0.723	0.719	0.735
Backpack dog + Bowl + Teapot	0.607	0.712	0.703	0.653	0.672	0.756	0.734	0.720	0.755	0.692	0.688	0.701
Backpack dog + White Cat + Emoji	0.648	0.703	0.717	0.674	0.692	0.812	0.719	0.741	0.701	0.742	0.720	0.796
Dog + Wolf + Backpack	0.717	0.738	0.722	0.547	0.565	0.552	0.679	0.681	0.707	0.766	0.795	0.831
Cat + Sunglasses + Boot	0.770	0.791	0.837	0.845	0.793	0.815	0.845	0.793	0.815	0.845	0.793	0.815
Table + Can + Teapot	0.836	0.827	0.810	0.753	0.770	0.741	0.751	0.799	0.806	0.818	0.771	0.829
Robot + Dog + Clock	0.663	0.638	0.693	0.689	0.764	0.797	0.645	0.674	0.710	0.661	0.715	0.717
Average	0.678	0.728	**0.759**	0.715	0.746	**0.783**	0.682	0.731	**0.756**	0.686	0.708	**0.732**

Results: V&L Domain

Results: V&L Domain

Experiments: NLP Domain

- Trained multiple LoRAs based on FLAN-T5 datasets (encoder-decoder)

- Tasks covered an array of topics that tested different specialties that different experts may cover
  - Translation, Natural Language Inference, Closed-Book Question Answering (Q&A)

- Different from V&L because each task is more aligned with a single LoRA adapter - shows a different benefit of the gating function

Results: NLP Domain

# Task	Metric	LoRAHub	PEMs	MoLE
Translation				
WMT '14 En→Fr	BLEU	27.4	25.6	**29.1**
WMT '14 Fr→En	BLEU	29.4	27.1	**31.3**
WMT '16 En→De	BLEU	24.6	24.9	**27.7**
WMT '16 De→En	BLEU	**29.9**	28.0	29.1
WMT '16 En→Ro	BLEU	17.7	15.2	**18.9**
WMT '16 Ro→En	BLEU	23.5	21.7	**25.1**
Average		25.4	24.2	**26.9**
Struct to Text				
CommonGen	Rouge-1	53.7	48.8	**55.1**
	Rouge-2	**23.1**	22.4	23.1
	Rouge-L	49.7	47.2	**53.9**
DART	Rouge-1	45.3	46.2	**48.8**
	Rouge-2	22.6	18.9	**23.5**
	Rouge-L	35.1	**37.6**	36.0
E2ENLG	Rouge-1	41.1	40.7	**42.0**
	Rouge-2	26.3	24.2	**29.0**
	Rouge-L	38.8	**42.1**	41.8
WebNLG	Rouge-1	52.1	52.0	**54.5**
	Rouge-2	23.9	24.6	**26.8**
	Rouge-L	45.2	47.8	**49.3**
Average		38.1	37.7	**40.3**

Closed-Book QA				
ARC-c	EM	51.7	50.4	**52.9**
ARC-e	EM	69.7	65.7	**70.3**
NQ	EM	17.3	16.1	**23.5**
TQA	EM	**54.5**	53.9	54.0
Average		48.3	46.5	**50.2**
Big-Bench Hard (BBH)				
Boolean Expressions	EM	55.1	53.0	**57.3**
Causal Judgement	EM	57.6	51.1	**57.9**
Date Understanding	EM	**31.0**	29.3	30.7
Disambiguation	EM	46.6	47.2	**49.3**
Penguins in a Table	EM	41.4	39.8	**45.0**
Reasoning Objects	EM	35.2	**37.5**	33.7
Ruin Names	EM	19.9	19.3	**21.2**
Average		38.4	33.2	**42.2**
Natural Language Inference (NLI)				
ANLI-R1	EM	81.0	80.3	**82.7**
ANLI-R2	EM	80.9	80.2	**82.4**
ANLI-R3	EM	77.4	76.6	**78.9**
QNLI	EM	77.6	78.0	**78.1**
Average		79.2	78.8	**80.5**

Analysis

- The balance loss component in MoLE improves entropy and encourages a more uniform composition weight distribution, enhancing performance
- Through this gating function, MoLE is able to preserve the individual components of LoRAs while fine tuning to specific tasks
- MoLE displays strong generalizability to new datasets compared to its counterparts displayed through NLI tasks
- Scales well as the number of LoRA adapters increases compared to other state of the art models

Limitations

- Although MoLE is lightweight, full parameter models will outperform it in certain domains

# Number of Concepts	Text-alignment					Average Image-alignment				
	NLA	Custom	Textual Inversion	SVDiff	MoLE	NLA	Custom	Textual Inversion	SVDiff	MoLE
3	0.678	<u>0.751</u>	0.709	0.728	**0.759**	0.694	**0.761**	0.720	0.719	<u>0.757</u>
4	0.681	**0.735**	0.721	0.717	<u>0.725</u>	0.712	**0.760**	0.736	0.721	<u>0.742</u>
5	0.652	<u>0.731</u>	0.704	0.723	**0.762**	0.682	**0.798**	0.710	0.708	<u>0.737</u>
6	0.678	0.722	**0.735**	0.709	<u>0.727</u>	0.698	0.721	**0.747**	0.712	<u>0.736</u>
Average	0.672	<u>0.734</u>	0.717	0.719	**0.752**	0.692	**0.760**	0.728	0.715	<u>0.743</u>

Limitations

- Generally strong with larger LoRAs but as the number increases to ~128 performance decreases for all models

# Number of LoRA	NLA	LoRAHub	PEMs	MoLE
8	32.7	33.9	33.7	**36.6**
24	36.8	37.1	36.9	**38.7**
48	34.4	36.9	34.6	**39.4**
128	34.1	35.5	34.9	**38.5**
Average	34.5	35.9	35.0	**38.3**

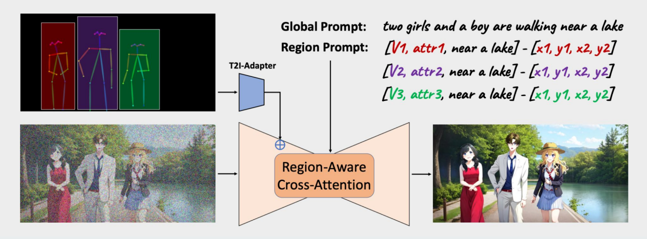
Thanks for watching!

Existing Methods: Reference tuning (V&L only)

- **Gradient Fusion** minimizes differences in **output activations,** optimizing the model to behave similarly to each LoRA

$$W = \arg\min_{W} \sum_{i=1}^{n} ||(W_0 + \Delta W_i)X_i - W X_i||_F^2$$

- **Controllable sampling** uses **position masks** to help the model learn how to apply LoRA modules effectively, allowing for **discriminate** generation

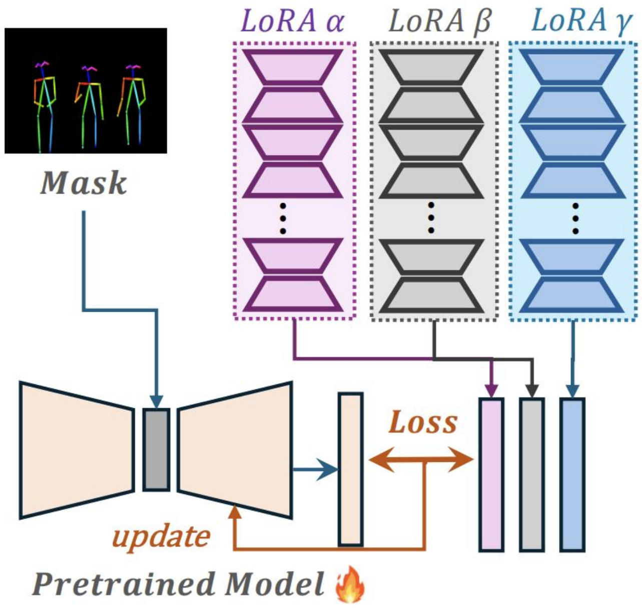


Issues: Reference Tuning

- Manual-designed masks = **compositional inflexibility**

- Need to retrain **entire model** by integrating outputs from multiple LoRA + position masks

- Substantial computational cost for retraining



Introduction: Mixture of LoRA Experts (MoLE)

Where should we put this gating function?

- **Observation:** Different layers of a LoRA module have **unique traits**, which collectively define the LoRAs overall attributes

- If we put a LoRA trained on combined dataset, we see that different layers specialize in handling different tasks

Adv. reasoning ← Simple QA

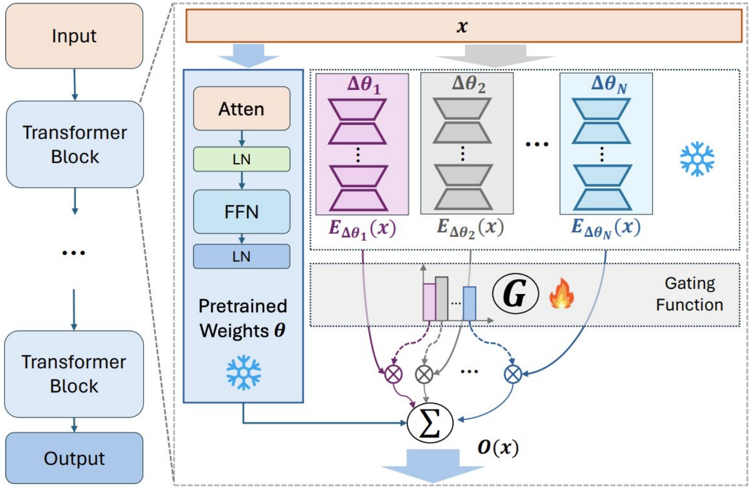
	ANLI-R1	ANLI-R2	QNLI
Full LoRA	81.65	80.03	76.42
0%-20%	78.72	78.35	**78.14**
20%-40%	76.10	77.96	77.85
40%-60%	76.95	**81.47**	74.57
60%-80%	77.25	78.19	75.71
80%-100%	**82.59**	77.91	75.48

Introduction: Mixture of LoRA Experts (MoLE)

Conclusion

- Since different layers can perform better on different tasks, **it is important to have layer-wise adaptation**

- **Let's have the gating function at each layer** to allow for finer grained specialization



Step 1: Get output of pretrained model block

$\boldsymbol{x} \in \mathbb{R}^{L \times d}$

Input

θ

Block parameters

$$\boldsymbol{x}'_\theta = \boldsymbol{x} + f_{\text{Attn}}\Big(\text{LN}(\boldsymbol{x})\,|\,\theta\Big)$$

Output of block
w/o LoRA

$$\boldsymbol{F}_\theta(\boldsymbol{x}) = \boldsymbol{x}'_\theta + f_{\text{FFN}}\Big(\text{LN}(\boldsymbol{x}'_\theta)\,|\,\theta\Big)$$

Step 2: Introduce LoRA experts

$\boldsymbol{x} \in \mathbb{R}^{L \times d}$

Input

θ

Block parameters

$\Omega = \{\Delta\theta_i\}_{i=0}^{N}$

LoRAs

$$\boldsymbol{x}'_{\Delta\theta_i} = \boldsymbol{x} + f_{\text{Attn}}\Big(\text{LN}(\boldsymbol{x})\big|\Delta\theta_i\Big)$$

LoRA Output

$$\boldsymbol{E}_{\Delta\theta_i}(\boldsymbol{x}) = \boldsymbol{x}'_{\Delta\theta_i} + f_{\text{FFN}}\Big(\text{LN}(\boldsymbol{x}'_{\Delta\theta_i})\big|\Delta\theta_i\Big)$$

Intuition: Finding how each LoRA $\Delta\theta$ affects transformer block's behavior **w/o** interference from the full pretrained weights.

Step 3: Apply gating function

Outputs from all LoRA experts are concatenated and normalized for stability:

$$\boldsymbol{E}_{\Omega}\left(\boldsymbol{x}\right) = \text{Normalization}\Big(\boldsymbol{E}_{\Delta\theta_0}\left(\boldsymbol{x}\right) \oplus \ldots \oplus \boldsymbol{E}_{\Delta\theta_{N-1}}\left(\boldsymbol{x}\right)\Big)$$

$$\boldsymbol{E}_{\Omega}\left(\boldsymbol{x}\right) \in \mathbb{R}^{\xi}$$

$$\xi = N \times L \times d$$

Next, flatten the outputs to N dimensions and multiply with our learnable parameter $\boldsymbol{e} \in \mathbb{R}^{\xi \times N}$

$$\varepsilon = \text{Flatten}\Big(\boldsymbol{E}_{\Omega}\left(\boldsymbol{x}\right)\Big)^{\top} \cdot \boldsymbol{e}$$

$$\varepsilon \in \mathbb{R}^{N}$$

Step 3: Apply gating function

Apply softmax (with learnable temperature т) to obtain the gating weights:

$$\mathcal{G}\left(\varepsilon_i\right) = \frac{\exp\left(\varepsilon_i/\tau\right)}{\sum_{j=1}^{N} \exp\left(\varepsilon_j/\tau\right)}$$

т determines whether to get more selective or balanced contributions from each LoRA

Step 3: Final Output

Find the output of gating function by multiplying LoRA output with gating probability

$$\tilde{\boldsymbol{E}}_{\Omega}(\boldsymbol{x}) = \sum_{i=0}^{N} \mathcal{G}_i\left(\varepsilon_i\right) \cdot \boldsymbol{E}_{\Delta\theta_i}\left(\boldsymbol{x}\right) \qquad \tilde{\boldsymbol{E}}_{\Omega}(\boldsymbol{x}) \in \mathbb{R}^{L \times d}$$

Now, final output of this block is the sum of pretrained network and gating function:

$$\boldsymbol{O}\left(\boldsymbol{x}\right) = \boldsymbol{F}_{\theta}\left(\boldsymbol{x}\right) + \tilde{\boldsymbol{E}}_{\Omega}\left(\boldsymbol{x}\right)$$

# Method	Text-alignment	Image-alignm	
		Concept 1	Concept 2

Appendix: Coarse Gating

# Method	Text-alignment	Image-alignment		
		Concept 1	Concept 2	Concept 3
m-MoLE	0.731	0.719	0.714	0.747
l-MoLE	0.760	0.727	0.731	**0.757**
b-MoLE	**0.766**	0.726	**0.737**	0.755
n-MoLE	0.722	**0.739**	0.682	0.730