

Model Customization & Instruction Tuning/LoRA

Aaditya Ghosalkar, Akira Durham, Ananya Ananda,
Sahlar Salehi, Daniel Slyepichev



Aaditya Ghosalkar
ag5jk

A large, abstract graphic on the left side of the slide features three overlapping circles in varying shades of blue and white. The circles are positioned in the upper left quadrant, creating a dynamic, modern look.

Low-Rank Adaptation for Foundation Models: A Comprehensive Review

Menglin Yang, Jialin Chen, Yifei Zhang, Jiahong Liu, Jiasheng Zhang, Qiyao Ma, Harshit Verma, Qianru Zhang, Min Zhou, Irwin King, Rex Ying

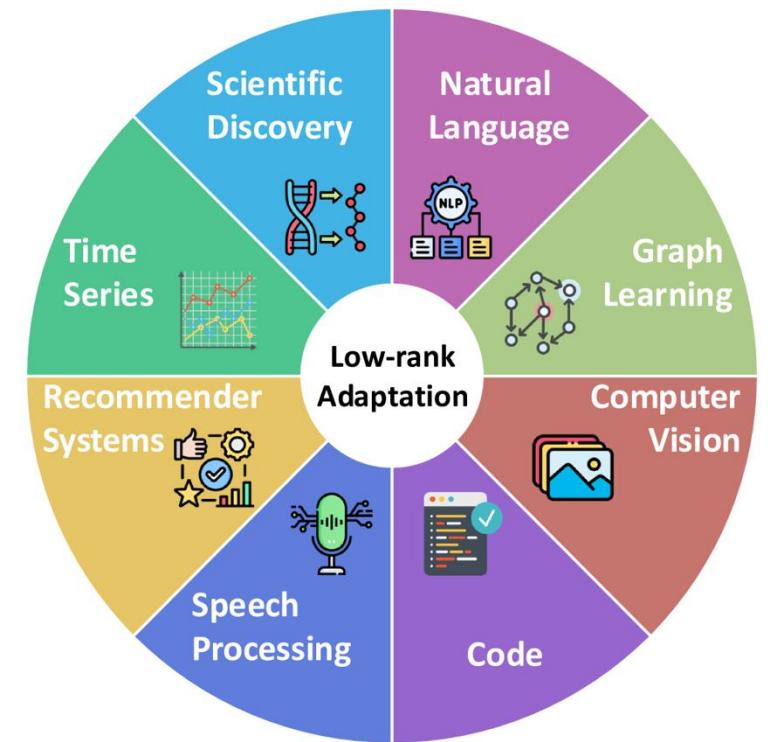
Foundation models

- Foundation models are large-scale neural networks mainly used for a variety of tasks.
 - Chat GPT
 - DeepSeek
 - LLama
- Not feasible to fine tune models with traditional methods
 - Updating all parameters (often billions – trillions of them)
 - Immense computational costs
 - Not super flexible for the variety of tasks needed for foundational modules

LoRA

- Low Rank Adaptation techniques involve implementing techniques that allow fine tuning with minimal overhead.
 - Allows models to be fine tuned for specific tasks without having to update all the parameters
 - Higher degree of customization

Works on two principles
fine tuning a low dimension subspace
training it using low-rank matrices

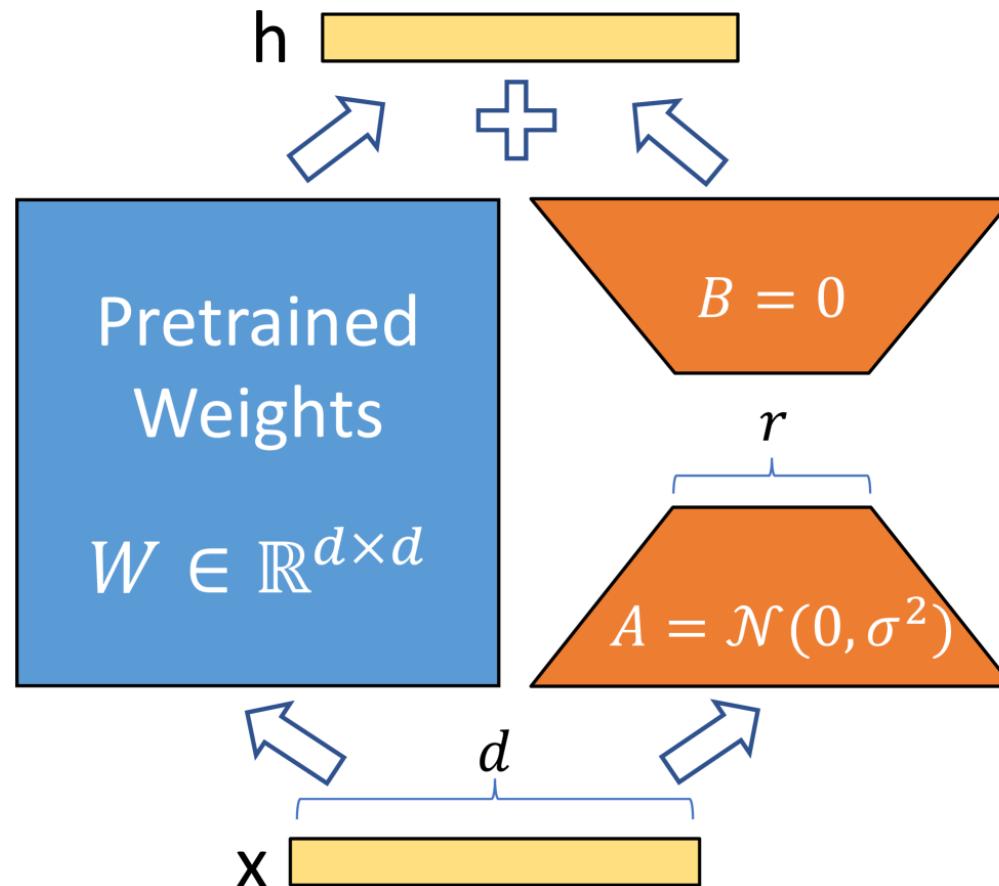


Basics

- Create a low-rank update matrix ΔW which is implemented through $\Delta W = BA\otimes$
- \otimes and A are which direction and how much you want to change each vector respectively
- This matrix is often fine tuned and added on over each cycle making more changed into the model
- α/r is a scaling factor controlling the magnitude of the low-rank update

$$f(x) = W_0x + \Delta Wx = W_0x + \frac{\alpha}{r}BAx,$$

Model of fine tuning process

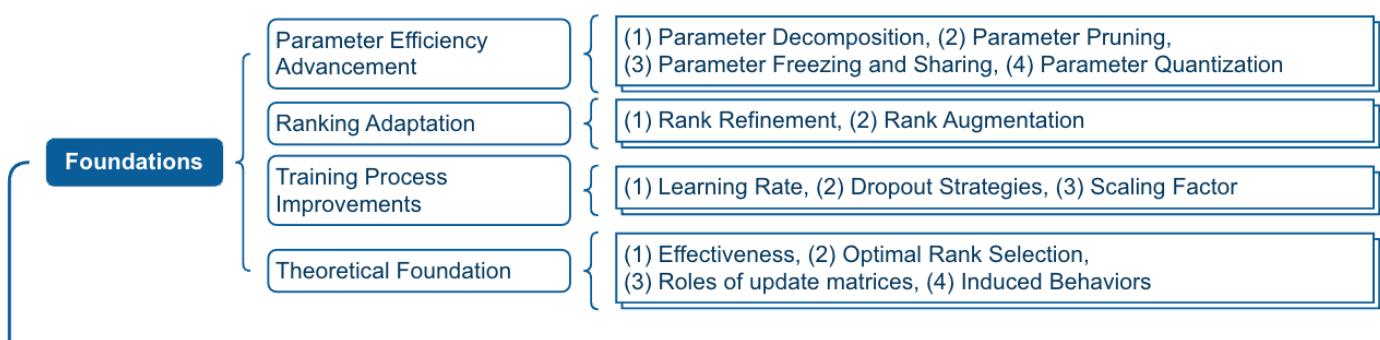


Advantages

- Drastically cuts down on the number of new parameters needed to train
- Focusing on the specific matrix leads to faster changes
- Does not slow down the model when using it
- Keeps the all-previous functionality since all original weights are frozen
- Easy to deploy for any scenario

Foundations for LoRA

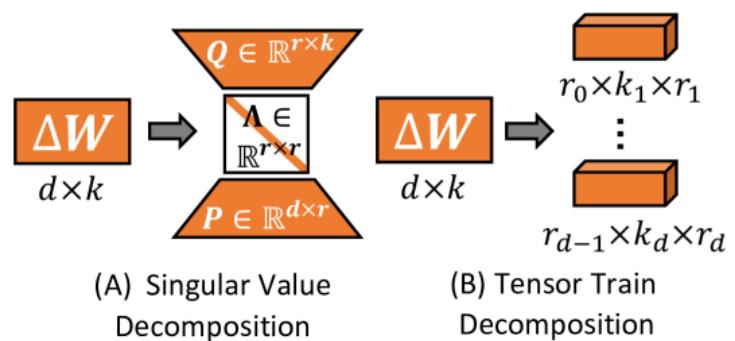
- Foundations
 - Looking into the foundational aspects of Lora and how they show their effectiveness of the process



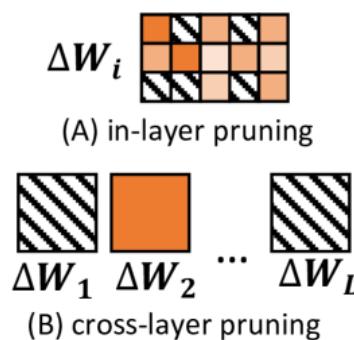
Parameter Efficiency Enhancement

4 Distinct Components that make up the process of making the process of training easier

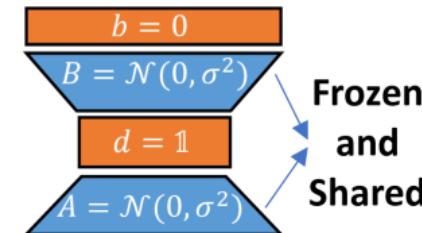
1. Parameter Decomposition



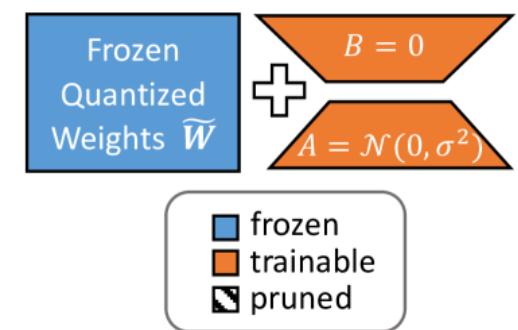
2. Parameter Pruning



3. Parameter Freezing and Sharing



4. Parameter Quantization



Parameter Decomposition

- Breaking down the learning process into smaller chunks to make it computationally cheaper and gives more control
- 2 main ways of doing this
 - Update Matrix Decomposition
 - Pre-trained Weight Decomposition

Update Matrix Decomposition

- Breaking down the ΔW matrix into other components, often times into smaller pieces with different properties to make for a more effective finetuning process

1. Parameter Decomposition

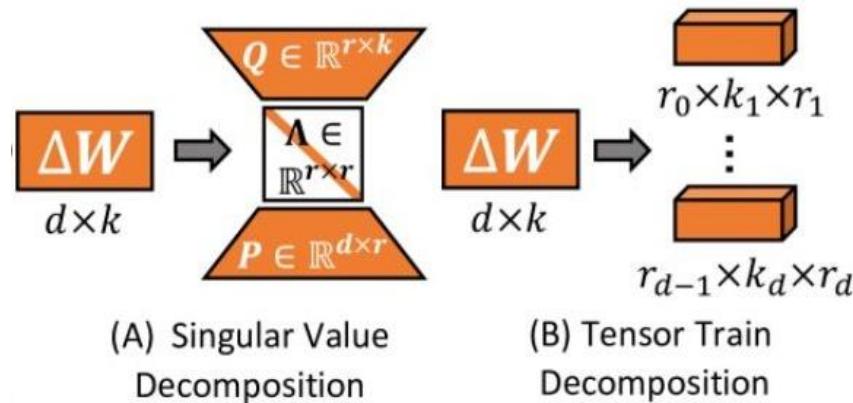


TABLE I: Summary of Weight Decomposition Methods for LoRA

LoRAs	Method	Decomposition Components
AdaLoRA [36]	SVD	$\Delta W: \Delta W \leftarrow P \Lambda Q^T$
BiLoRA [37]	SVD	$\Delta W: \Delta W \leftarrow P \Lambda Q^T$
LoRETTA_{rep} [34]	Tensor Train	$\Delta W: \Delta W \leftarrow \text{TT}(B) \cdot \text{TT}(A)$
LoRETTA_{adp} [34]	Tensor Train	$\Delta W: \Delta W \leftarrow \text{TT}(\Delta W) // \text{series}$
TT-LoRA [38]	Tensor Train	$\Delta W: \Delta W \leftarrow \text{TT}(\Delta W) // \text{parallel}$
DoRA [39]	Normalization	$W_0: W_0 \leftarrow \ W_0\ _c \frac{W_0}{\ W_0\ _c}$

Pre-trained Weight Decomposition

Breaks down the pretrained weight W_0 into magnitude and directional components.

This breaks down the original weights into a more flexible format meaning you can change specific vectors of the original weights instead of the whole thing

$$W_0 = m \frac{V}{\|V\|_c} = \|W_0\|_c \frac{W_0}{\|W_0\|_c},$$

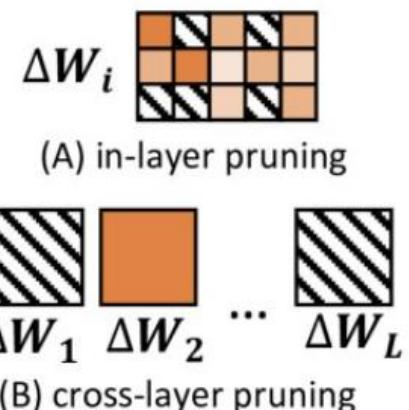
$$W' = m \frac{W_0 + BA}{\|W_0 + BA\|_c},$$

Parameter Pruning

- Selectively removing less important parameters
 - Importance based
 - Assigning importance to each parameter based on criteria and removing the ones under a specific threshold
 - Regularization-based
 - Removing parameters by assigning penalty terms, that rewards it for assigning zero to any parameters, parameters that are zero are considered pruned
 - Output-based
 - Getting rid of parameters who don't have much of an effect on the model's behavior

Method	Strategy	Mechanism	Core Innovation
<i>Parameter Pruning</i>			
SparseAdapter [42]	Importance-based	Parameter scoring	Multi-criteria importance evaluation (magnitude, gradient, sensitivity)
SoRA [43]	Regularization-based	Gated sparsification	L1-regularized gating for adaptive sparsity
LoRA-Drop [44]	Output-based	Layer impact analysis	Dynamic pruning based on layer-wise output contributions

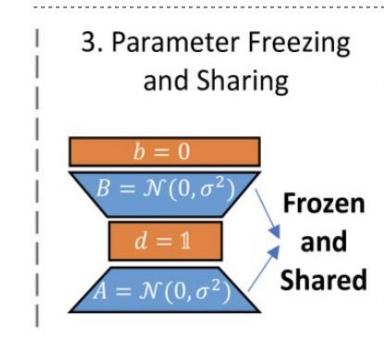
2. Parameter Pruning



Parameter Freezing

- Freezing one of the LoRA matrices (either A or B) so that the other can be trained independently
 - It is possible to freeze random vectors in matrix A while only updating matrix B
 - Because matrix A works like an extractor and B acts like a projector, so only changing B is an option

Parameter Freezing				
LoRA-FA [45]	Selective freezing	Fixed feature extraction	A	Random initialization and freezing of matrix A
Asymmetric LoRA [46]	Theoretical design	Orthogonal projection	B	Random orthogonal A with theoretical guarantees



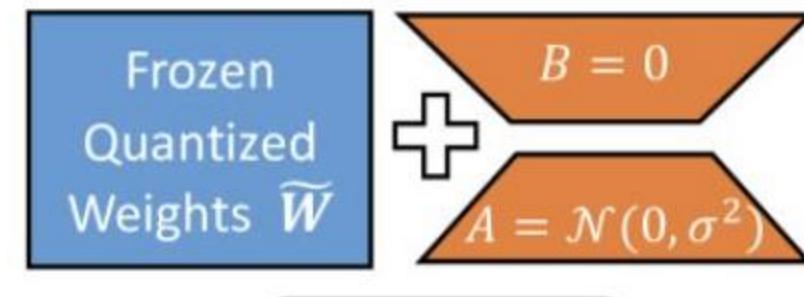
Parameter Quantization

Quantization optimizes a model by lowering the bit at which the parameters are stored at.

For Lora, you can quantize the training matrices to speed up the process of training.

- Quantization Timing
 - Can be done before, during, or after finetuning
- Quantization Technique
 - Uniform, Non-uniform, and mixed-precision

4. Parameter Quantization

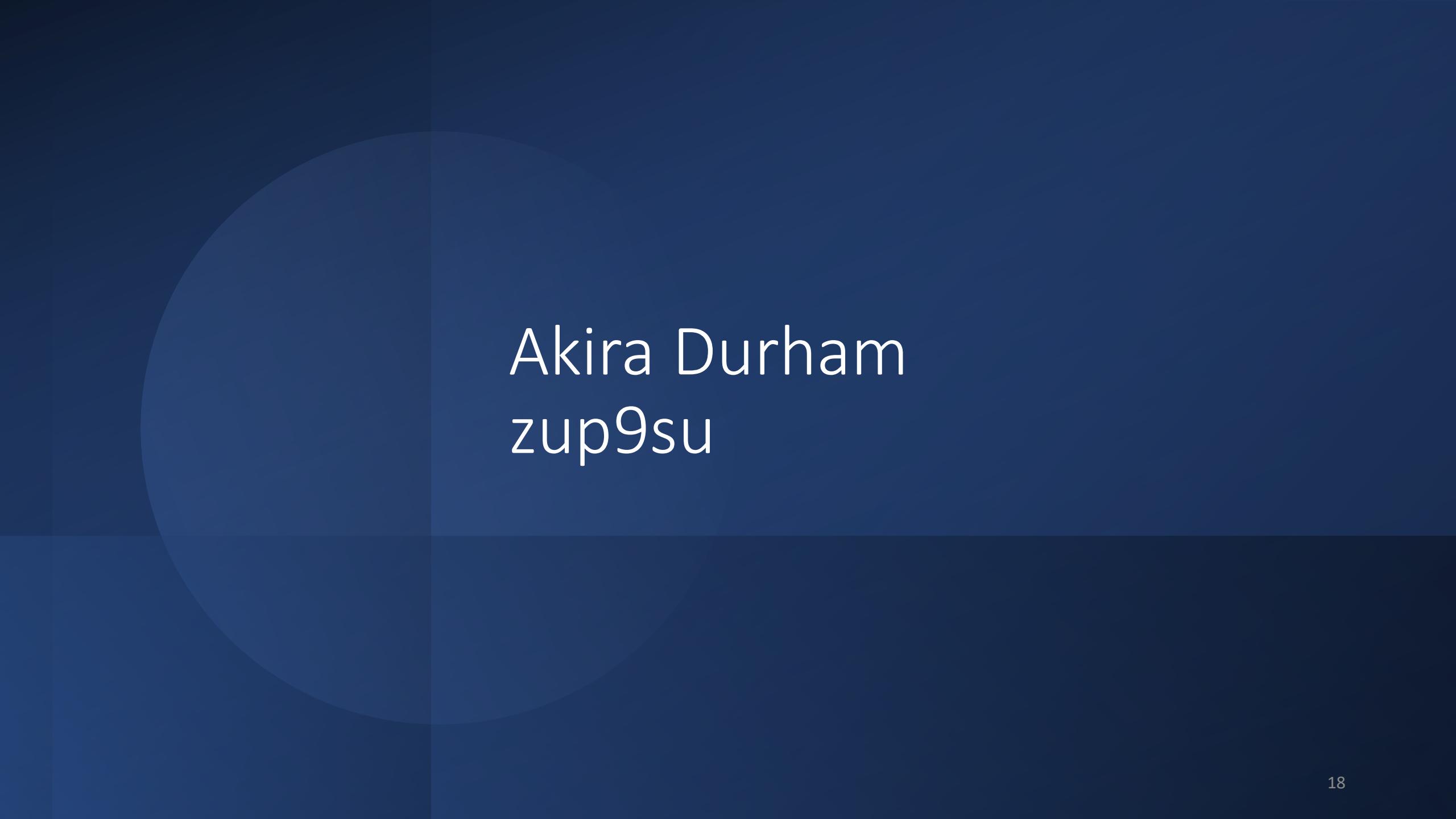


LoRA Quantization

TABLE III: Comparison of Quantization Methods for LoRA

Method	Timing	Target	Precision	Technique	Low-Rank	Optimization	Memory Focus	Dequant
QLoRA [55]	Pre-FT	Pretrained	4 bit	NormalFloat	Standard LoRA	Separate	FT&Inference	Partial
QA-LoRA [57]	Pre & During FT	Pretrained	2, 3, 4 bit	Group-wise	Q-aware LoRA	Joint	FT&Inference	None
LoftQ [56]	Pre-FT	Pretrained	mixed	Uniform & NormalFloat	Q-aware LoRA	Joint	FT	Partial
LQER [58]	Post-FT	Pretrained	mixed	Group-wise & adaptive	SVD-based LR	Q-error min	Inference	None
QDyLoRA	During FT	Pretrained	mixed	Rank sampling	Dynamic LoRA	Rank selection	FT	None
LQ-LoRA [59]	Pre-FT	Pretrained	mixed	ILP & data-aware	Q-aware LoRA	Joint	FT&Inference	Partial

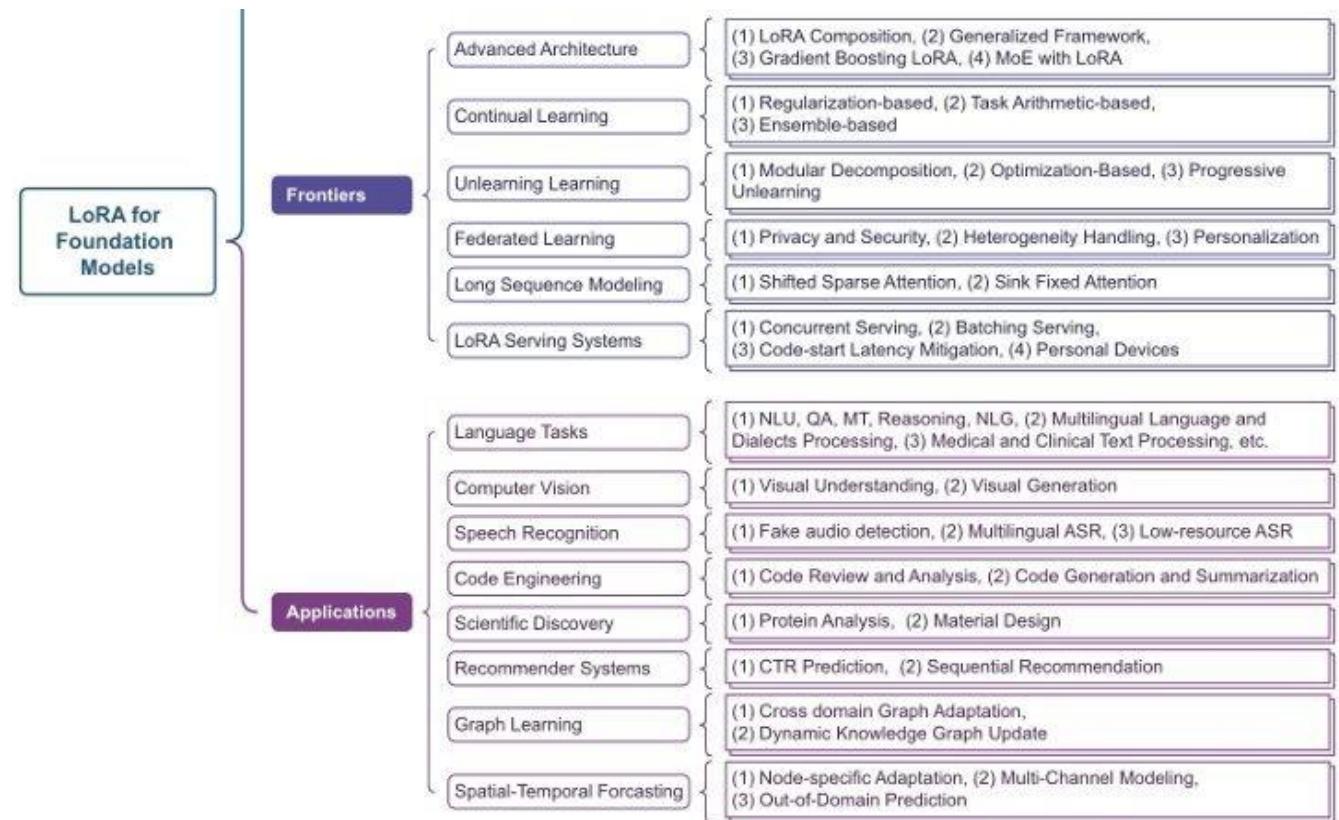
FT = Fine-tuning, Q = Quantization, LR = Low-Rank, ILP = Integer Linear Programming

The background features a dark blue gradient with three semi-transparent white circles of varying sizes. One large circle is positioned in the upper left, another medium-sized circle is in the lower center, and a smaller circle is in the lower left.

Akira Durham
zup9su

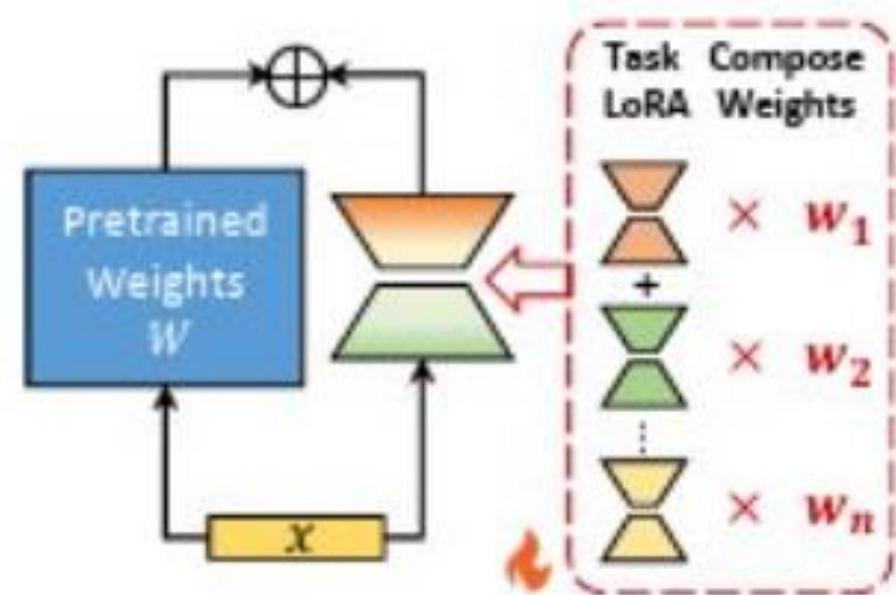
Frontiers in LoRA and Applications

- Frontier developments that extend LoRA capability
 - Enable new functionalities
 - Handle more complex tasks
 - Address challenges in adaptation
- Application
 - LoRA adaptation across domains
- Challenges and Discussion
 - Critical challenges
 - Opportunities for investigation
- Conclusion



Advanced Architecture

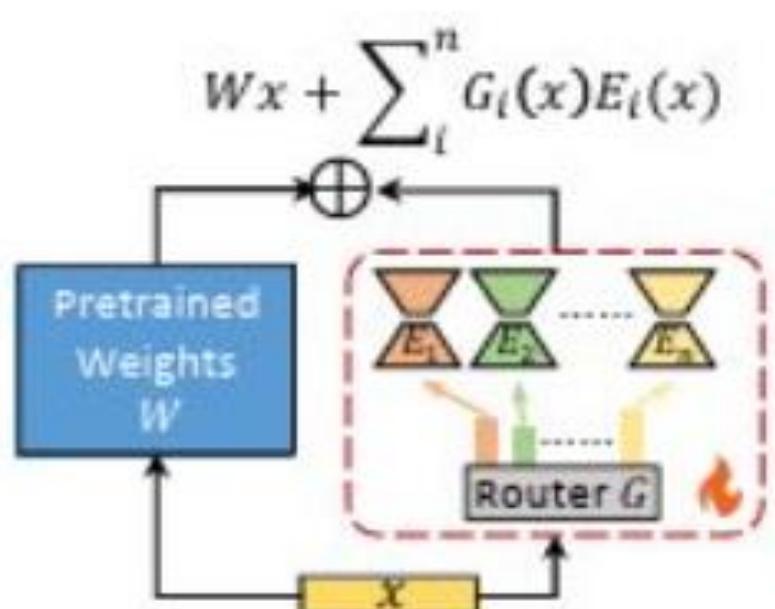
- Goals: Improve LoRA for performance, parameter efficiency, generalization
- LoRA Composition
 - Dynamic composition of multiple LoRA modules to enhance adaptability + generalization
 - Optimization-based, Retrieval-based, Batch-oriented Compositions
- Generalized Framework
 - Extending LoRA architecture to capture both task-specific and general features better
 - Dual-branch and Multi-PEFT United Frameworks



(A) Composition of LoRA Modules

Advanced Architecture

- Mixture of Experts
 - Multiple "expert" sub-networks specializing in different input patterns
 - Gating mechanism routes inputs to network experts, allowing for broad input handling
 - Efficiency-oriented Design, Memory-based Adaptation, Task-based Integration



(B) Mixture of LoRA Experts

$$y = \sum_i^n G_i(x)E_i(x) \quad (13)$$

where y is the output, G_i is a gating function, E_i is an expert, and n is the number of experts.

TABLE 6
Comparison of MoE-LoRA Methods

Method	Routing	Key Feature
MoV/MoLoRA [84]	Soft routing	All experts contribute with weights
MoELoRA [85]	Top-k	Contrastive between experts
MoLA [89]	Top-k	Layer-wise expert distribution
LoRAMoE [90]	Top-k	Localized balancing for knowledge
MoRAL [91]	Soft routing	Lifelong learning framework
MOELoRA [92]	Task-based	Task identifier conditioning
MoCLE [93]	Cluster-based	Instruction cluster routing
LLaVA-MoLE [94]	Token-level	Top-1 sparse expert selection

LoRA for Learning and Unlearning

- Continual Learning
 - Efficient nature of LoRA allows for updating models without forgetting
 - Reduced cost vs. fine-tuning, isolation of task-specific knowledge, flexible combinations
 - Regularization-based, Task arithmetic-based, Ensemble-based techniques
- Unlearning
 - Targeted removal of specific knowledge from models without retraining
 - Modular Decomposition, Optimization-Based, Sequential Pipeline Strategies

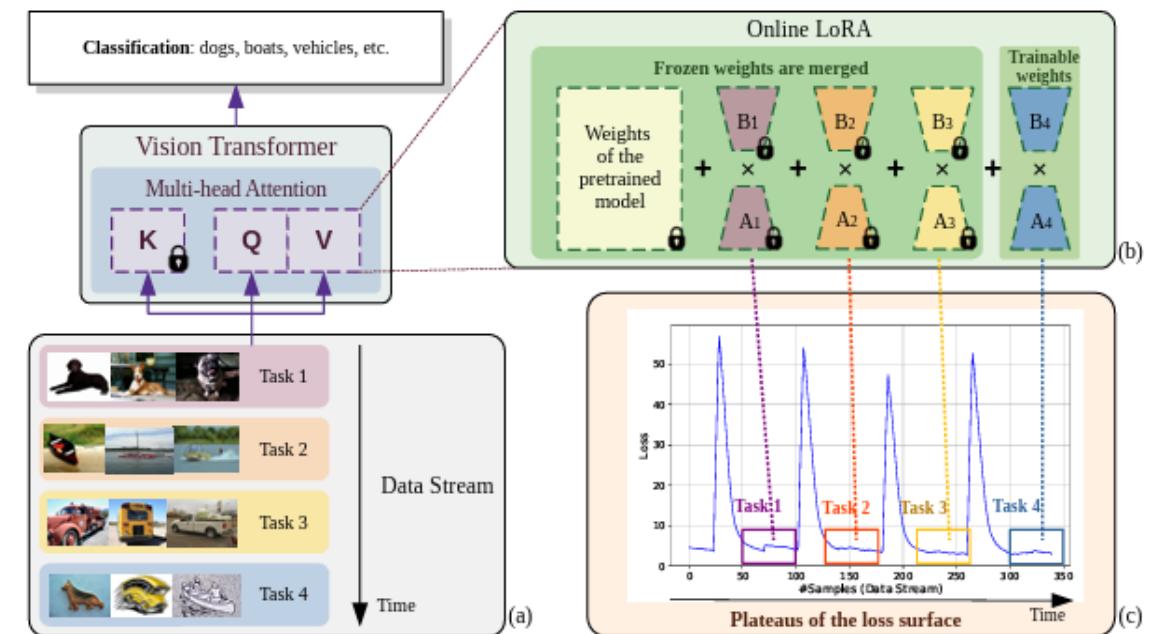
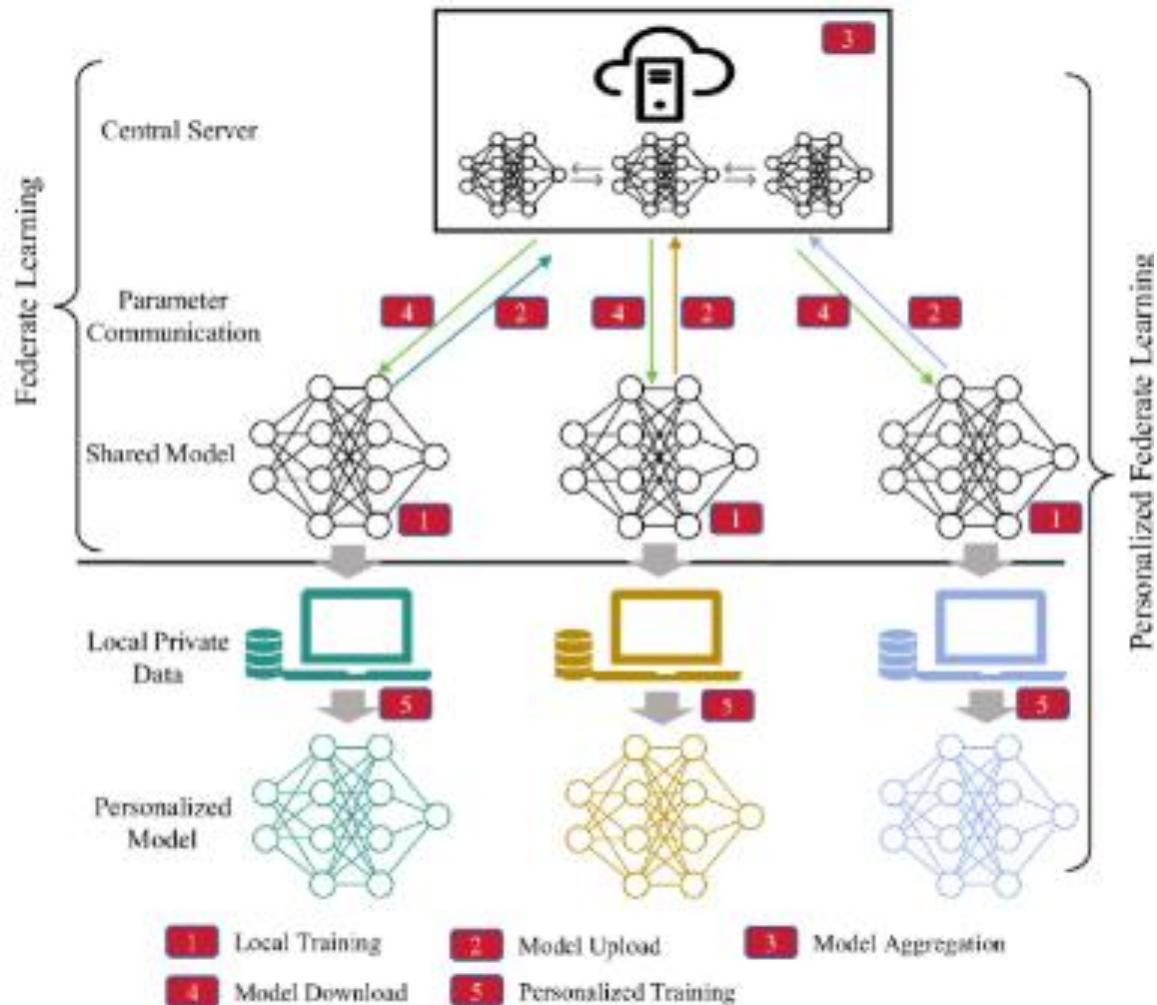


Figure 1. The overview of Online-LoRA. As the data is continuously streamed (a), a new pair of trainable LoRA parameters (A_4, B_4) is added (b) every time the loss surface encounters a plateau (c). Subsequently, the previous LoRA parameters ($A_1, B_1; A_2, B_2; A_3, B_3$) are frozen (the lock sign in (b)) and merged to the weights of the pre-trained ViT model.

Federated Learning

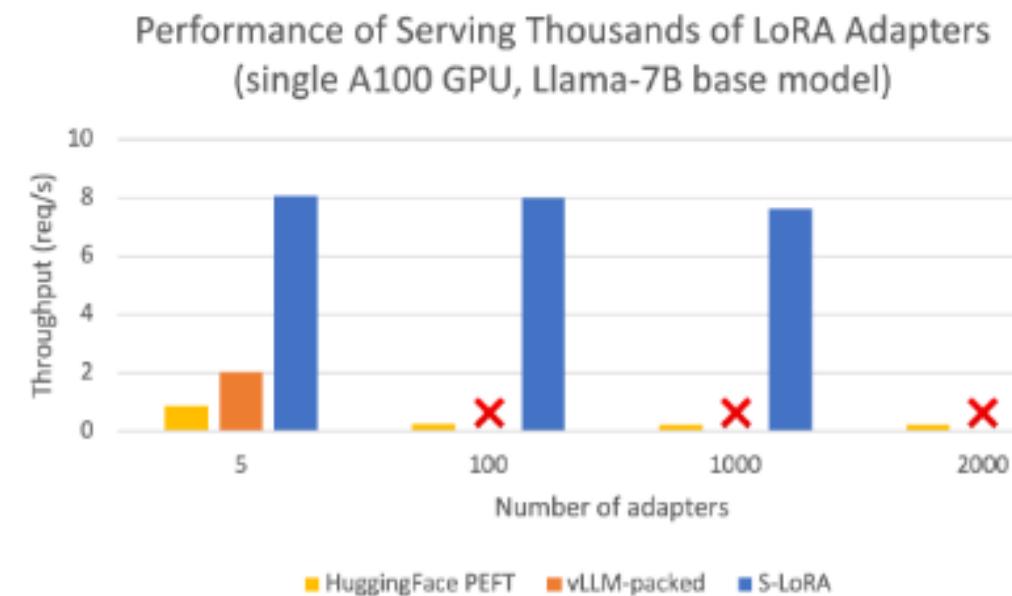


- **Federated Learning**

- Leverage collective knowledge while maintaining robust protection for individual data
- Increased foundation models availability to resource-constrained devices, IoT
- Privacy and Security, Computation Efficiency, Heterogeneity Handling, Personalization

Long Sequence and Serving

- Long Sequence Modeling
 - Extend context window of foundation models, typically constrained by max context length due to quadratic computational complexity of self-attention w.r.t sequence length
 - Shifted Sparse Attention and Sink Fixed Attention methods
- LoRA Serving Systems
 - Efficient serving of multiple LoRA models is essential – GPU management, cold-start, etc
 - Paging mechanism for KV cache and LoRA weights, CPU-assisted, scheduling algorithm



Applications I

- Language Tasks
 - Application in NLP to improve inference latency and training times
 - Multilingual Language + Dialects and Medical + Clinical Text
- Computer Vision
 - Enhance adaptability across visual tasks including understanding and generation
 - Visual Understanding
 - Visual Generation
- Speech Recognition



(a) Zero-shot Grounding results on data #38 in Aircraft dataset [9].

V-LoRA LMM

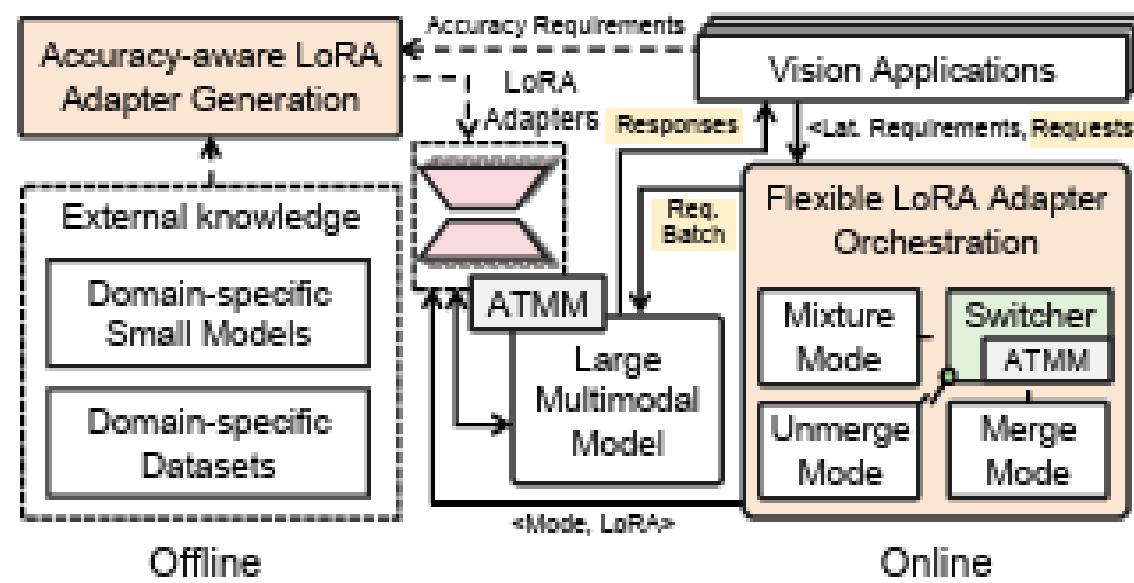


Figure 8. VaLoRA overview.

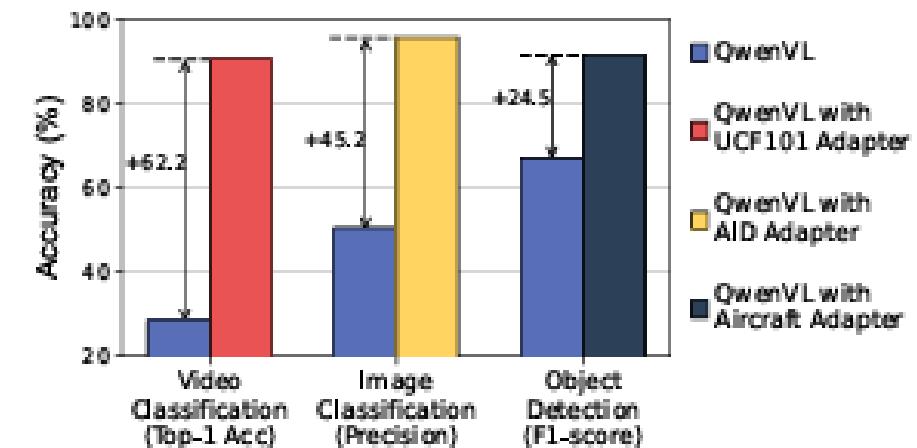


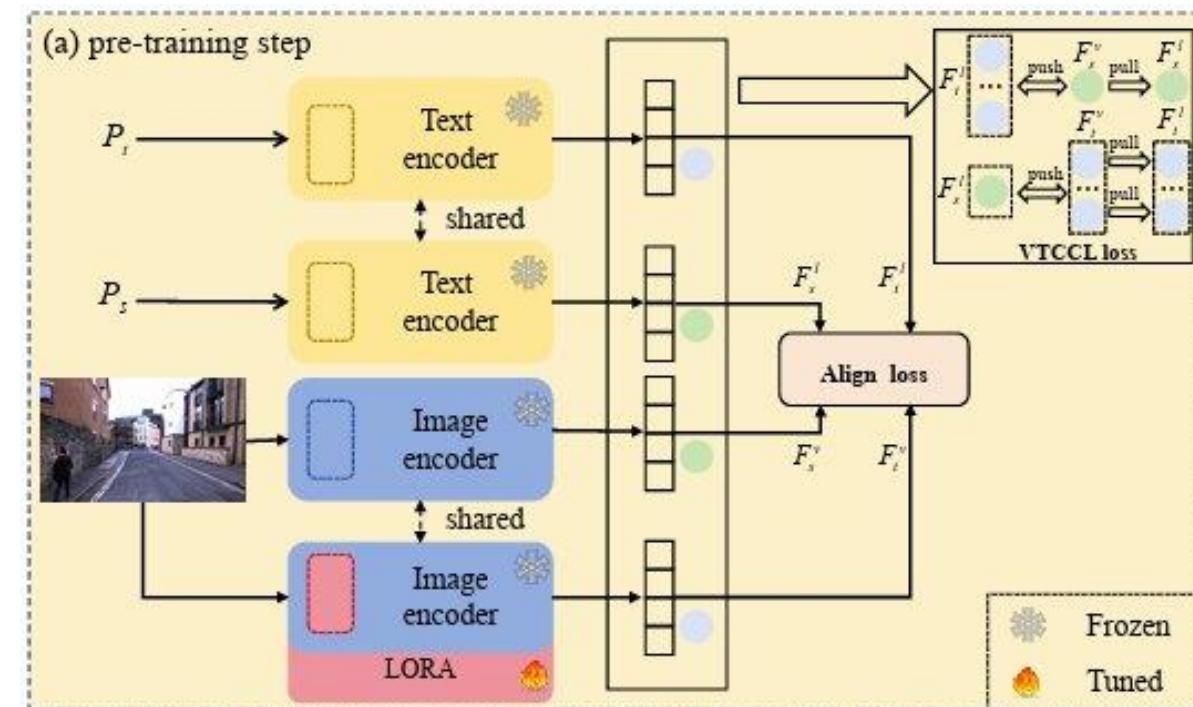
Figure 4. LoRA adapters with domain-specific knowledge improve the Qwen-VL's accuracy on target tasks.

Applications II

- Code Engineering
 - Review, Analysis, Generation, Summarization
- Scientific Discovery
 - Protein Analysis and General Science Area
- Recommender Systems
 - Click-Through Rate Prediction and Sequential Recommendation
- Graph NN Learning
 - Non-Euclidean data fine-tuning to adapt new graphs or structure updates
 - Cross-domain Graph NN Adaptation and Dynamic Knowledge Graph Learning

Applications III

- Spatial-Temporal Forecasting
 - LoRA to address multivariate time series data
 - Node-specific Adaptation, Multi-channel Modeling, Out-of-domain Prediction
- Multi-Modal
 - MFM combine different data modalities -> optimize training efficiency + alignment
 - Language-vision Learning
 - Language-audio Learning



MMD-LoRA

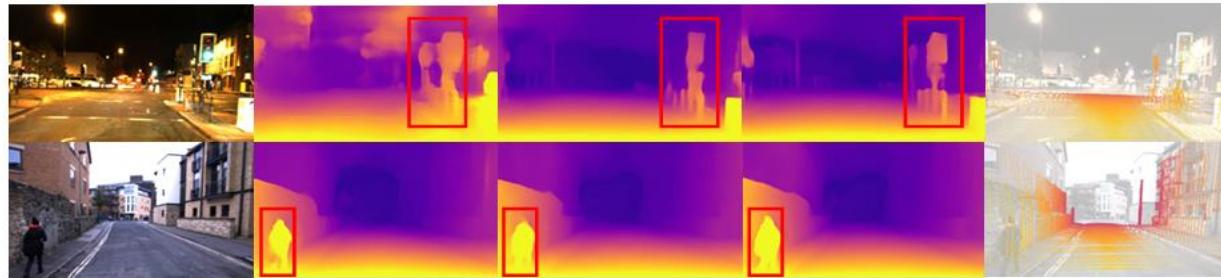


Fig. 4. Qualitative results of MMD-LoRA and the previous SOTA depth estimation method on the Robotcar test set. The first column denotes the original image. The second, third and fourth column denote the depth estimation results of Monodepth2, md4allDD and ours MMD-LoRA respectively. The final column indicates the ground-truth depth maps.

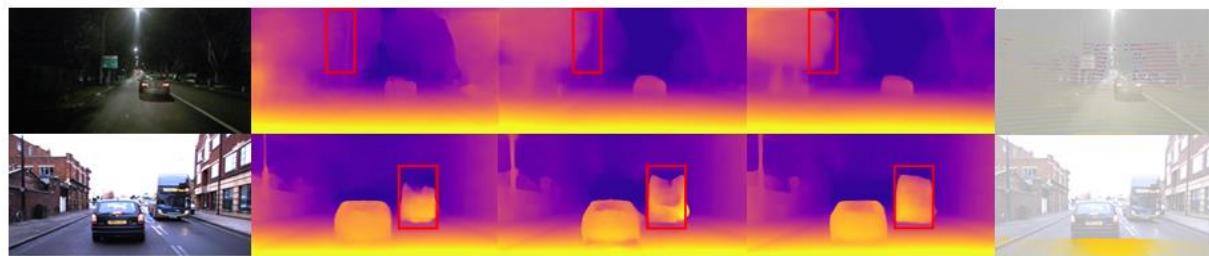
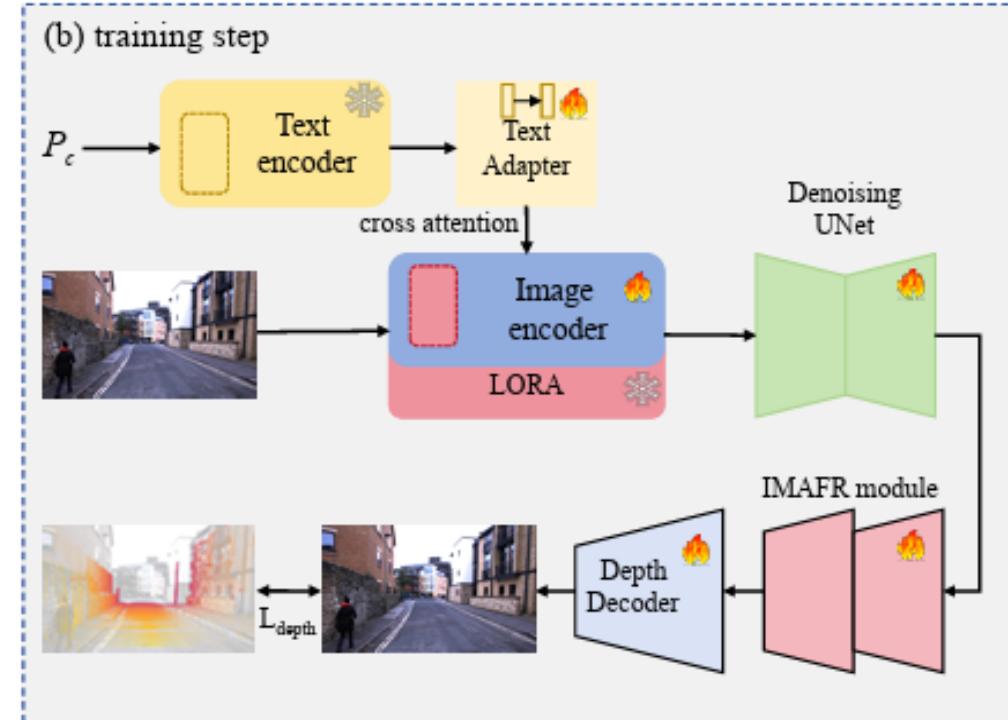
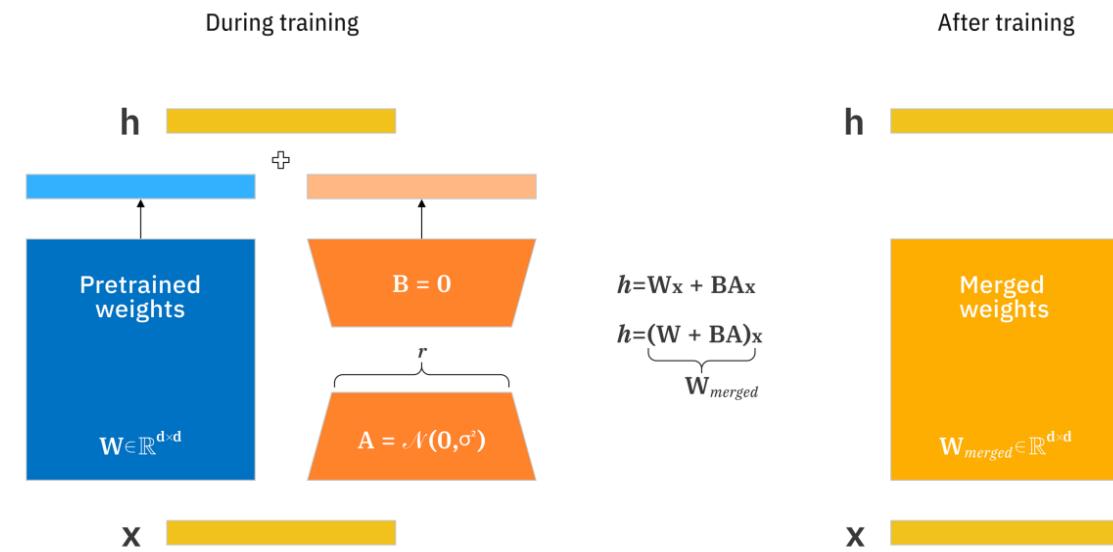


Fig. 5. Ablation visualization of our proposed MMD-LoRA with PDDA and VTCCL. The first column denotes the original image. The second, third and fourth column denote baseline(*i.e.* EVP [4]), MMD-LoRA with PDDA, MMD-LoRA with PDDA and VTCCL. The final column indicates the ground-truth depth maps.



Challenges and Future Directions

- Challenges
 - Theoretical Understanding
 - Architectural Design Principles
 - Computation Efficiency
 - Robustness and Verification
 - Privacy and Security





DoRA:

Weight-Decomposed Low-Rank Adaptation



Ananya Ananda
jaf5rp

Motivation

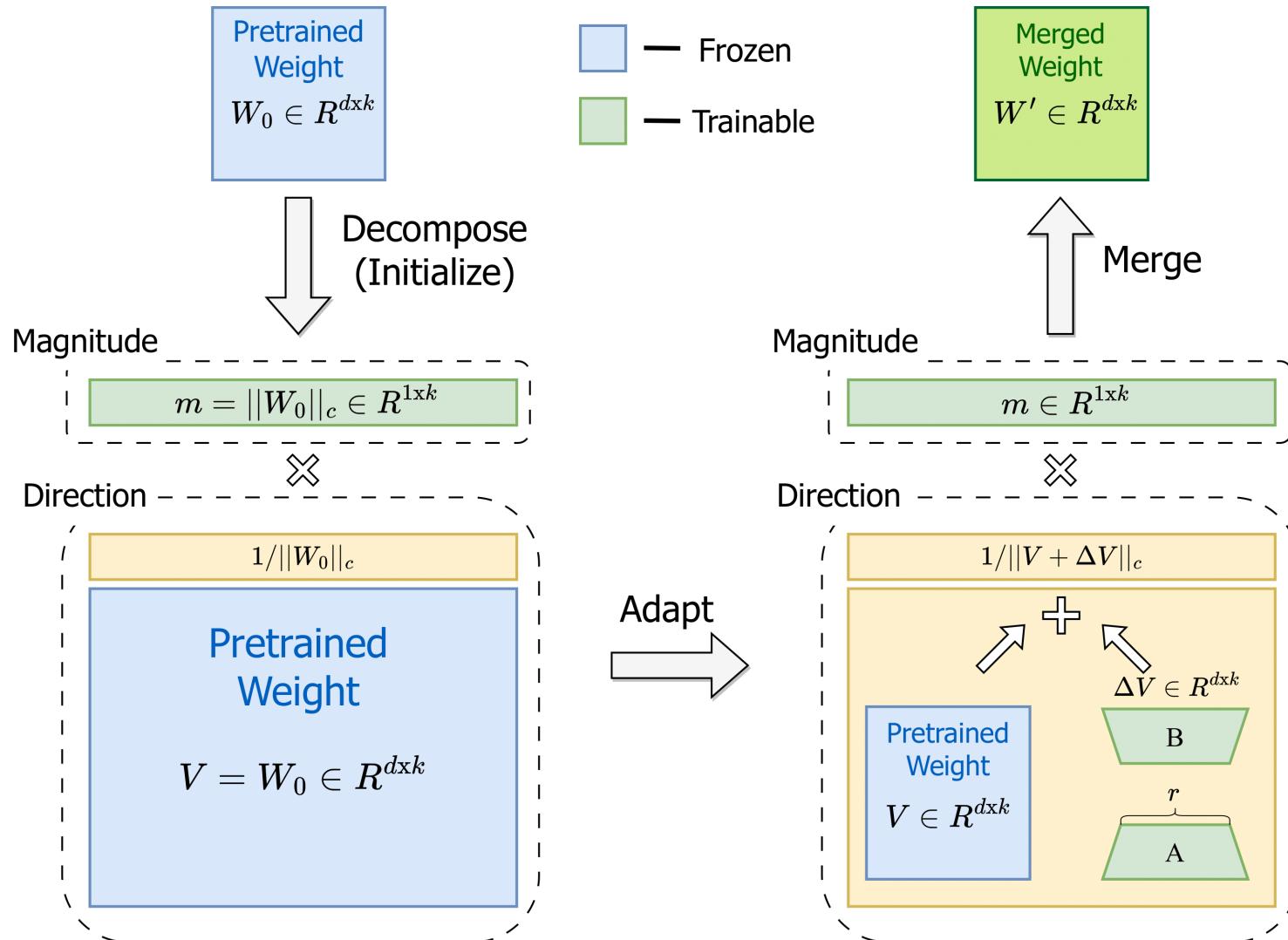
- LoRA Limitations
 - Accuracy gap between LoRA & fine-tuning (FT)
 - Limited trainable parameters may restrict learning capacity
 - No clear understanding of how LoRA differs from FT
- **DoRA (Weight-Decomposed Lower Rank Adaptation)**
 - Enhance both learning capacity and training stability of LoRA while avoiding additional inference overhead
 - Aim to resemble learning capacity of fine-tuning
 - Inspired by weight normalization

Weight Normalization

- **Weight normalization** - achieves faster convergence via improving conditioning of gradient w/ weight reparameterization
- Decomposes weights into 2 components: **magnitude** (g) & **direction** (\mathbf{v})
 - Stabilizes training
 - Achieve faster convergence
 - Improved robustness

$$\mathbf{w} = \frac{g}{\|\mathbf{v}\|} \mathbf{v}$$

DoRA Overview



- Decomposes pre-trained weight into 2 components: **magnitude & direction** for fine-tuning
- LoRA to update direction component

LoRA

- LoRA proposes using product of two-rank matrices to update the pre-trained weights incrementally

$$W' = W_0 + \Delta W = W_0 + \underline{BA} \quad (1)$$

Weight Decomposition Analysis

- Break weights down into 2 components:
 - **Magnitude** – how large the weight is
 - **Direction** – orientation of weight in space

$$W = m \frac{V}{\|V\|_c} = \|W\|_c \frac{W}{\|W\|_c} \quad (2)$$

Weight Decomposition Analysis

- Magnitude difference (3) and directional difference (4) between the pre-trained weight and fine-tuned weight

$$\Delta M_{\text{FT}}^t = \frac{\sum_{n=1}^k |m_{\text{FT}}^{n,t} - m_0^n|}{k} \quad (3)$$

$$\Delta D_{\text{FT}}^t = \frac{\sum_{n=1}^k (1 - \cos(V_{\text{FT}}^{n,t}, W_0^n))}{k} \quad (4)$$

Weight Decomposition Analysis

- Examine updates in magnitude and direction of LoRA and fine-tuning weights relative to pre-trained weights
 - Distinction between FT & LoRA likely mirror respective learning capability

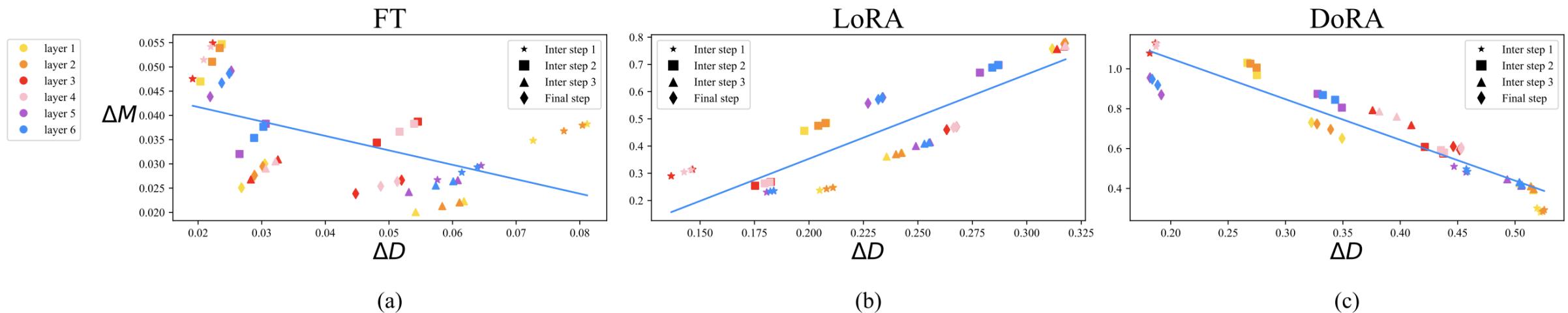
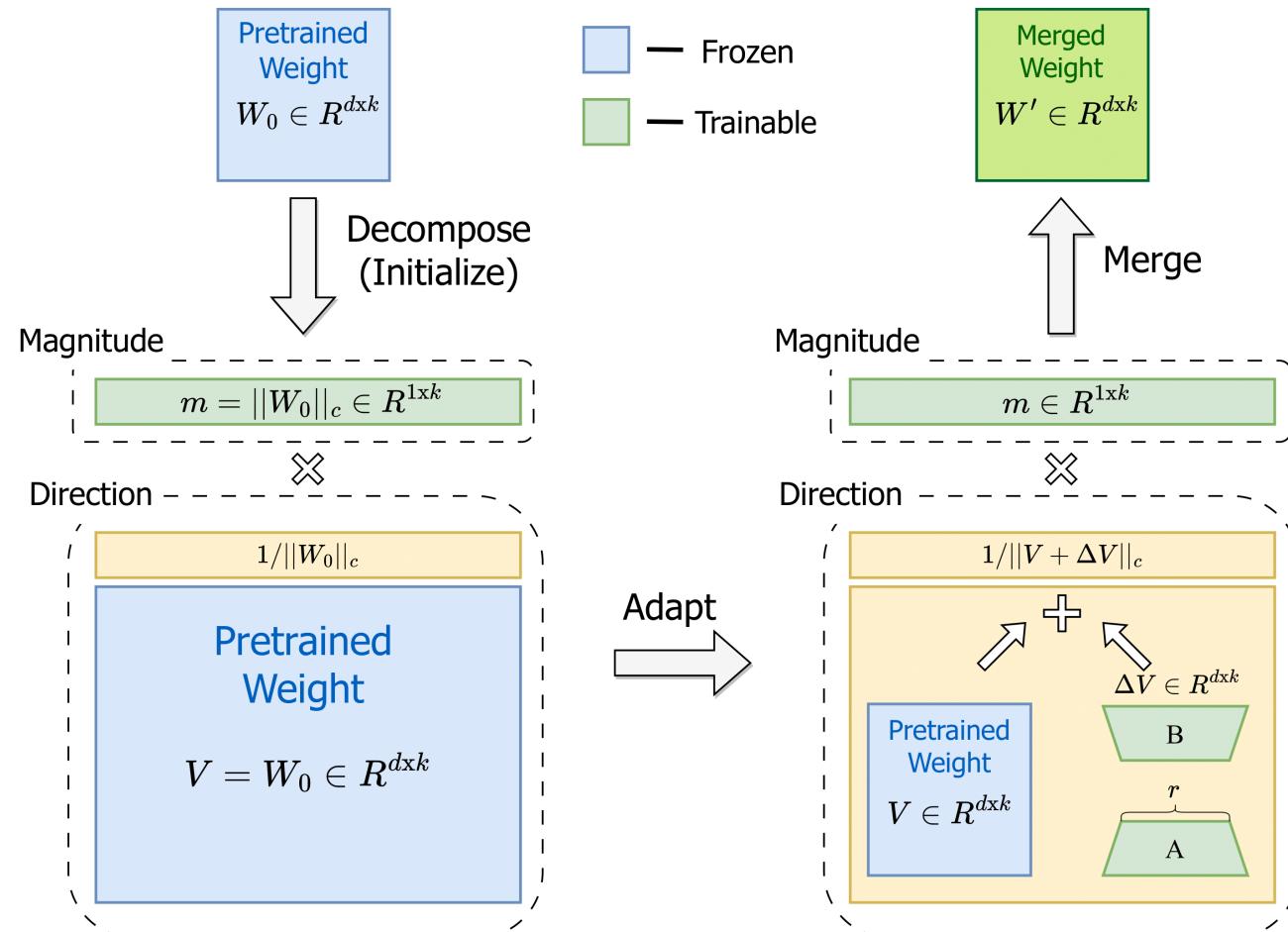


Figure A.1: Weight decomposition analysis on the value weight matrix

Weight-Decomposed Low-Rank Adaptation



$$W' = \frac{V + \Delta V}{\|V + \Delta V\|_c} = \frac{W_0 + BA}{\|W_0 + BA\|_c}$$

- Keep V frozen, but fine-tune magnitude and LoRA-style directional updates



Sahlar Salehi
rmh7ce

Gradient Analysis of DoRA

- Decomposition enhances learning stability of LoRA

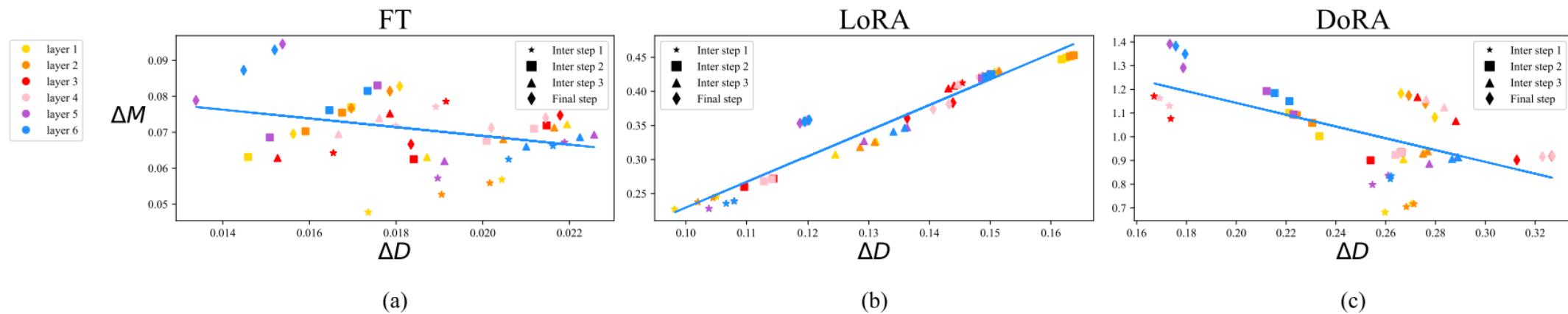
$$\nabla_{V'} \mathcal{L} = \frac{m}{\|V'\|_c} \left(I - \frac{V' V'^T}{\|V'\|_c^2} \right) \nabla_{W'} \mathcal{L} \quad (6)$$

Gradient Analysis of DoRA

- Gives insight into learning patterns of DoRA
- Smaller directional update=> larger magnitude update, **aligns with pattern of FT**

$$\nabla_m \mathcal{L} = \frac{\nabla_{W'} \mathcal{L} \cdot V'}{\|V'\|_c} \quad (7)$$

FT vs LoRA vs DoRA: Slope Comparison



DoRA Training Optimizations

- Unlike LoRA, gradient of low-rank updates differs from that of $W' \Rightarrow$ extra memory requirements
- Solution: redefine gradient of loss w.r.t directional matrix
- Reduced gradient graph memory consumption, negligible change in accuracy
 - 24.4% memory reduction in fine tuning LLaMA, 12.4% for VL-BART

$$\nabla_{V'} \mathcal{L} = \frac{m}{C} \nabla_{W'} \mathcal{L} \text{ where } C = \|V'\|_c$$

Evaluating DoRA

- Compared DoRA with LoRA, FT, and various PEFT Methods
- Evaluate accuracy with respect to:
 - Common sense reasoning tasks
 - Image/video-text understanding tasks
 - Visual instruction tuning tasks
 - Test DoRA with LoRA variants
 - Adjusting rank settings
 - Change in tuning granularity

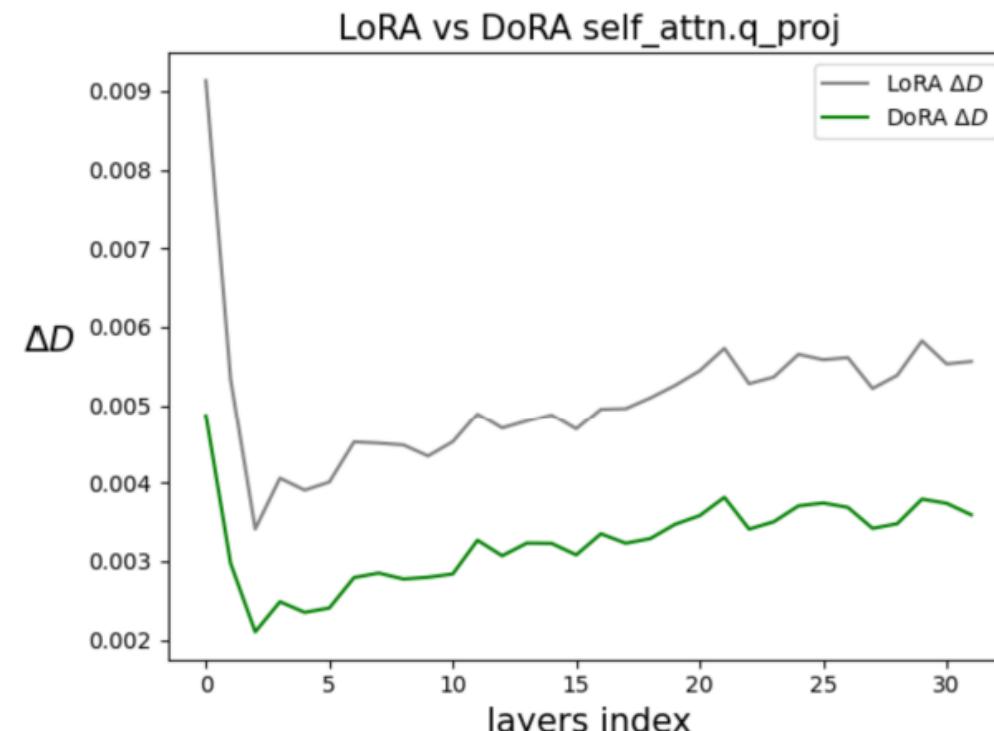
Common Sense Reasoning Tasks

Model	PEFT Method	# Params (%)	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
ChatGPT	-	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA-7B	Prefix	0.11	64.3	76.8	73.9	42.1	72.1	72.9	54.0	60.6	64.6
	Series	0.99	63.0	79.2	76.3	67.9	75.7	74.5	57.1	72.4	70.8
	Parallel	3.54	67.9	76.4	78.8	69.8	78.9	73.7	57.3	75.2	72.2
	LoRA	0.83	68.9	80.7	77.4	78.1	78.8	77.8	61.3	74.8	74.7
	DoRA [†] (Ours)	0.43	70.0	82.6	79.7	83.2	80.6	80.6	65.4	77.6	77.5
	DoRA (Ours)	0.84	69.7	83.4	78.6	87.2	81.0	81.9	66.2	79.2	78.4
LLaMA-13B	Prefix	0.03	65.3	75.4	72.1	55.2	68.6	79.5	62.9	68.0	68.4
	Series	0.80	71.8	83	79.2	88.1	82.4	82.5	67.3	81.8	79.5
	Parallel	2.89	72.5	84.9	79.8	92.1	84.7	84.2	71.2	82.4	81.4
	LoRA	0.67	72.1	83.5	80.5	90.5	83.7	82.8	68.3	82.4	80.5
	DoRA [†] (Ours)	0.35	72.5	85.3	79.9	90.1	82.9	82.7	69.7	83.6	80.8
	DoRA (Ours)	0.68	72.4	84.9	81.5	92.4	84.2	84.2	69.6	82.8	81.5
LLaMA2-7B	LoRA	0.83	69.8	79.9	79.5	83.6	82.6	79.8	64.7	81.0	77.6
	DoRA [†] (Ours)	0.43	72.0	83.1	79.9	89.1	83.0	84.5	71.0	81.2	80.5
	DoRA (Ours)	0.84	71.8	83.7	76.0	89.1	82.6	83.7	68.2	82.4	79.7
LLaMA3-8B	LoRA	0.70	70.8	85.2	79.9	91.7	84.3	84.2	71.2	79.0	80.8
	DoRA [†] (Ours)	0.35	74.5	88.8	80.3	95.5	84.7	90.1	79.1	87.2	85.0
	DoRA (Ours)	0.71	74.6	89.3	79.9	95.5	85.6	90.5	80.4	85.8	85.2

DoRA/LoRA Deviation from Pre-Trained Weights



(a)



(b)

Image/Video-Text Understanding Tasks

Method	# Params (%)	VQA^{v2}	GQA	NVLR²	COCO Cap	Avg.
FT	100	66.9	56.7	73.7	112.0	77.3
LoRA	5.93	65.2	53.6	71.9	115.3	76.5
DoRA (Ours)	5.96	65.8	54.7	73.1	115.9	77.4

Method	# Params (%)	TVQA	How2QA	TVC	YC2C	Avg.
FT	100	76.3	73.9	45.7	154	87.5
LoRA	5.17	75.5	72.9	44.6	140.9	83.5
DoRA (Ours)	5.19	76.3	74.1	45.8	145.4	85.4

Visual Instruction Tuning

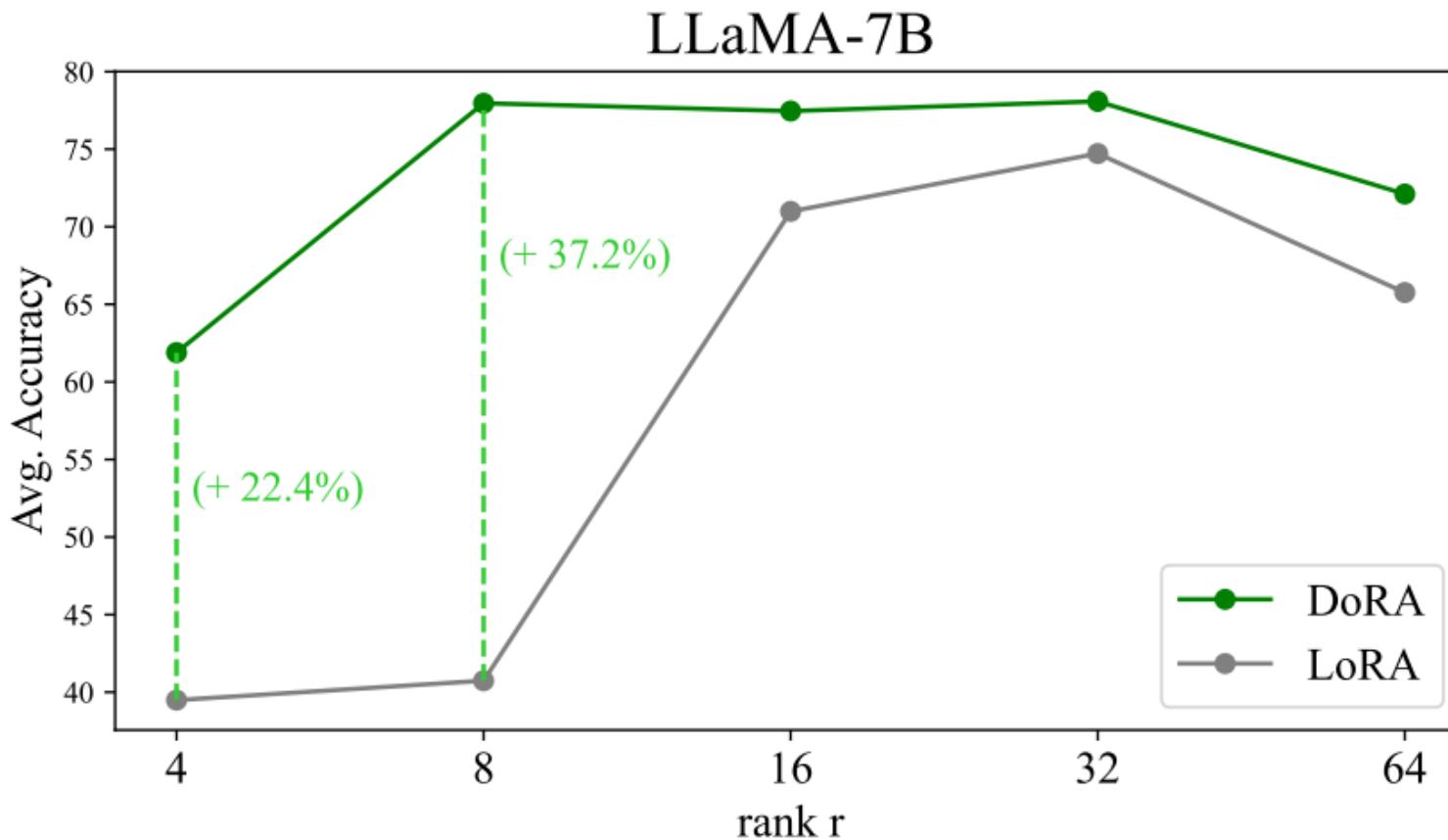
Method	# Params(%)	Avg.
FT	100	66.5
LoRA	4.61	66.9
DoRA (Ours)	4.63	67.6

Compatibility of DoRA with VeRA: DVoRA

- Vector-based Random Matrix Adaptation (VeRA)
 - Reduces trainable parameters
- DVoRA: DoRA + VeRA for directional update

Model	PEFT Method	# Params (%)	Score
LLaMA-7B	LoRA	2.31	5.1
	DoRA (Ours)	2.33	5.5
	VeRA	0.02	4.3
	DVoRA (Ours)	0.04	5.0
LLaMA2-7B	LoRA	2.31	5.7
	DoRA (Ours)	2.33	6.0
	VeRA	0.02	5.5
	DVoRA (Ours)	0.04	6.0

DoRA vs LoRA: Rank Configuration



DoRA vs LoRA: Granularity Analysis

- Idea: use **fewer trainable parameters** for directional updates
 - Update direction only for QKV modules

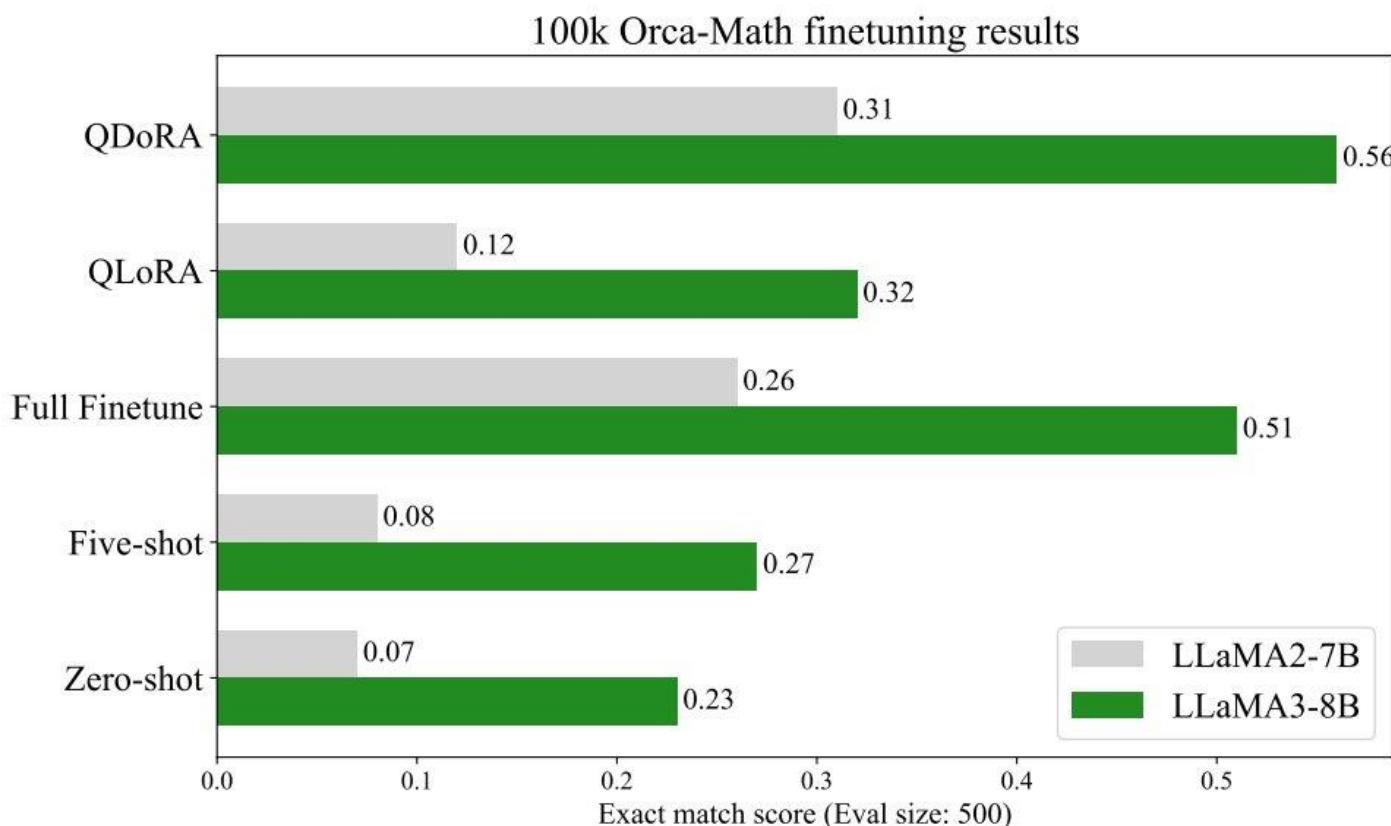
(Q)uery, (K)ey, (V)alue, (O)utput, (G)ate, (U)p, (D)own.

Model	PEFT Method	# Params (%)	m	v	Avg.
LLaMA-7B	LoRA	0.83	-	-	74.7
	DoRA (Ours)	0.84	QKVUD	QKVUD	78.1
	DoRA (Ours)	0.39	QKVOGUD	QKV	77.5
LLaMA-13B	LoRA	0.67	-	-	80.5
	DoRA (Ours)	0.68	QKVUD	QKVUD	81.5
	DoRA (Ours)	0.31	QKVOGUD	QKV	81.3

QDoRA: DoRA + QLoRA

- Problem: A lot of GPU memory needed to initially load model weights
- QLoRA: quantize pretrained model to 4-bit, fine tune with LoRA on top
- QDoRA: Use QLoRA for directional update
- Outperformed QLoRA and FT using much less memory

QDoRA Results

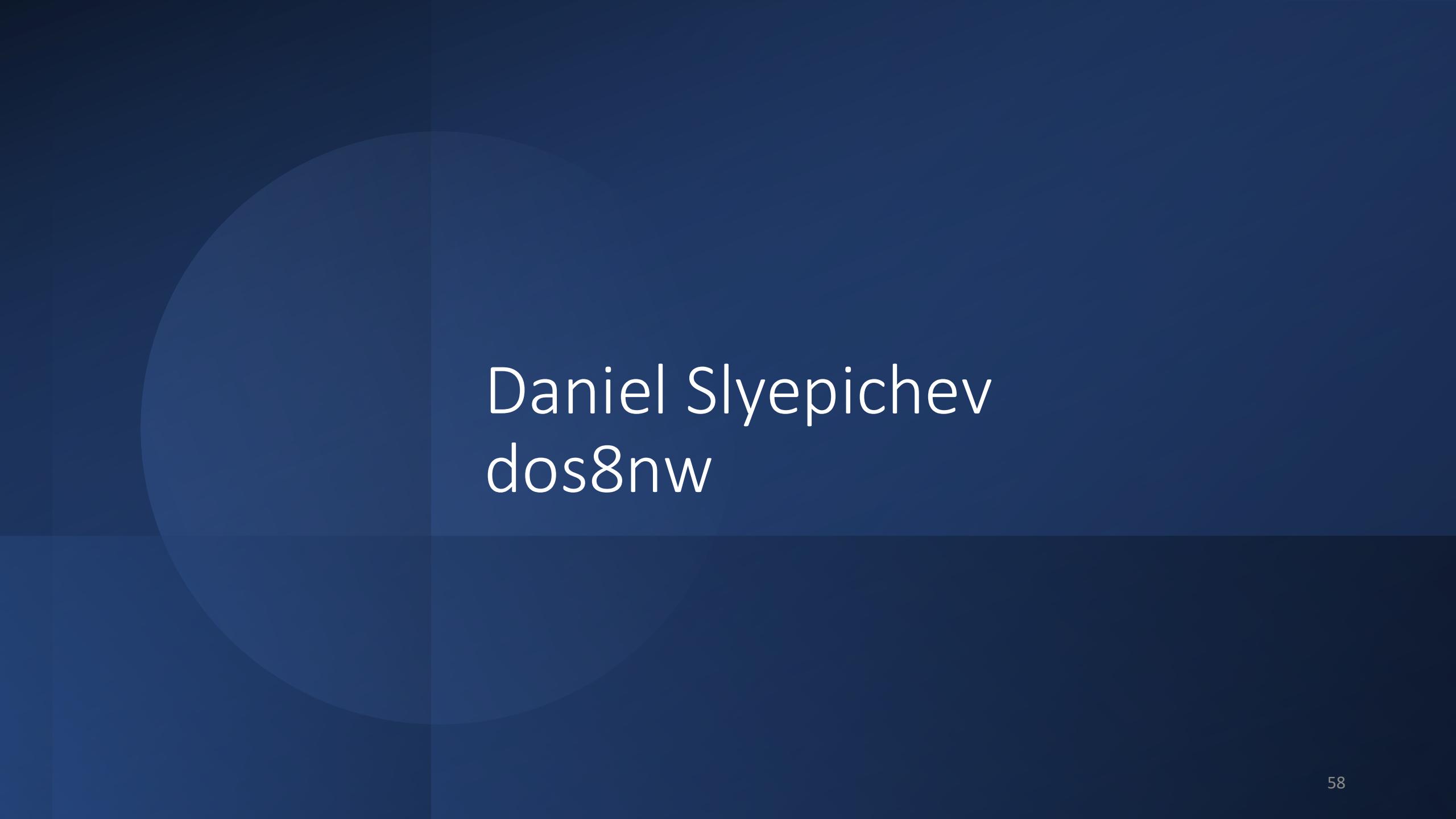


Further Applications: Text-to-Image Generation

- Compared DoRA vs LoRA on image generation
 - Same configuration/hyperparameters, test on generating 3D icons and Lego images
- Results: DoRA much more personalized than LoRA with same settings
 - Ex: training data for Lego dataset had Lego logo, DoRA outputs consistently added Lego logo to generated image where LoRA may not have
- Future directions with DoRA: look more into audio generation/data modes

Key Takeaways

- DoRA splits weight matrix into magnitude and directional components
- Fewer parameters than FT, more accurate than LoRA
 - Best of both worlds
- Can be optimized to use less GPU memory (QDoRA)
- Improvements over existing fine tuning for image generation

The background features a dark blue gradient with three semi-transparent white circles of varying sizes. One circle is positioned in the upper left quadrant, another in the lower left quadrant, and a third, larger one is centered in the upper right quadrant.

Daniel Slyepichev
dos8nw



SFT Memorizes, RL Generalizes: A Comparative Study of Foundation Model Post-training

Introduction

- Supervised Fine Tuning (SFT) & Reinforcement Learning (RL)
 - Generalizable, or just memorization?
- How will we study this question?
 - Text Based Rules
 - Model's ability to apply learned rules (given text instructions) to variants of those rules
 - Visualization
 - Performance consistency to variations in visual input, such as color and spatial layout
- What tasks will we solve?
 - General Points
 - Given 4 playing cards, find sum of target number
 - V-IRL
 - Navigation Task



Related Works

- Post-Training
 - Supervised Fine Tuning
 - A "Format Teacher" for training pre-trained LLMs
 - Weight Adjustment
 - Reinforcement Learning
 - Used for solving a specific task or alignment
 - Rewards system
- Memorization & Generalization
 - LLMs exhibit more overfitting on simpler, knowledge-intensive tasks and greater generalization on more complex, reasoning-intensive ones
 - Pre-computing reasoning graphs before autoregressive generation leads to generalization

Related Works

- Inference Time Computation
- VLM Perception Issues
 - Can be improved with more visual encoders, high quality SFT data, or unfreezing the visual backbone

Ground Truth Cards: [2, 8, 5, J]
Error Type: Fail to recognize all numbers

```
{  
  "cards": [10, 10, 5, 9],  
  "formula": "10+",  
  "thoughts": "'10+' is an incomplete formula, since '10+10-5+9=24', I should append '10' to the  
  current formula",  
  "action": "10"  
}
```

Evaluation Task: General Points

System Prompt (v_0^{in})

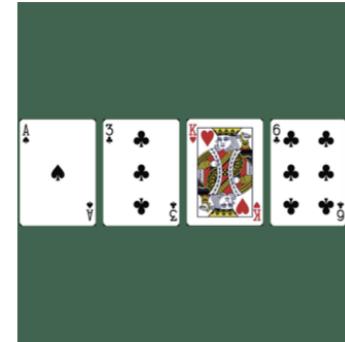
[Task Description]

You are an expert 24 points card game player. You are observing these four cards in the image. Note that 'J', 'Q', and 'K' count as '10', and each card must be used once. Your goal is to output a formula that evaluates to 24 using numbers from the cards and operators such as '+', '-', '*', '/', '(', ')', and '='.

[Output]

Your response should be a valid json file in the following format:

```
{  
  "cards": [x, y, z, w], where 'J', 'Q', and 'K' count as '10',  
  "number": [a, b, c, d], where a, b, c, and d are the numbers on the cards,  
  "formula": "an equation that equals 24",  
}
```



Appending model and verifier outputs to obtain v_t^{in}

$$v_t^{\text{in}} = [v_0^{\text{out}}, v_0^{\text{ver}}, v_1^{\text{out}}, v_1^{\text{ver}}, \dots, v_{t-1}^{\text{out}}, v_{t-1}^{\text{ver}}]$$

$$\triangleright v_t^{\text{in}} = \text{concat}(v_0^{\text{in}}, [v_k^{\text{out}}, v_k^{\text{ver}}]_{k=0}^{t-1})$$

Model output (v_t^{out})

```
{  
  "cards": ['A', '3', 'K', '6'],  
  "number": [1, 3, 13, 6],  
  "formula": "(1+6)*3+13=24",  
}
```

Verifier Output (v_t^{ver})

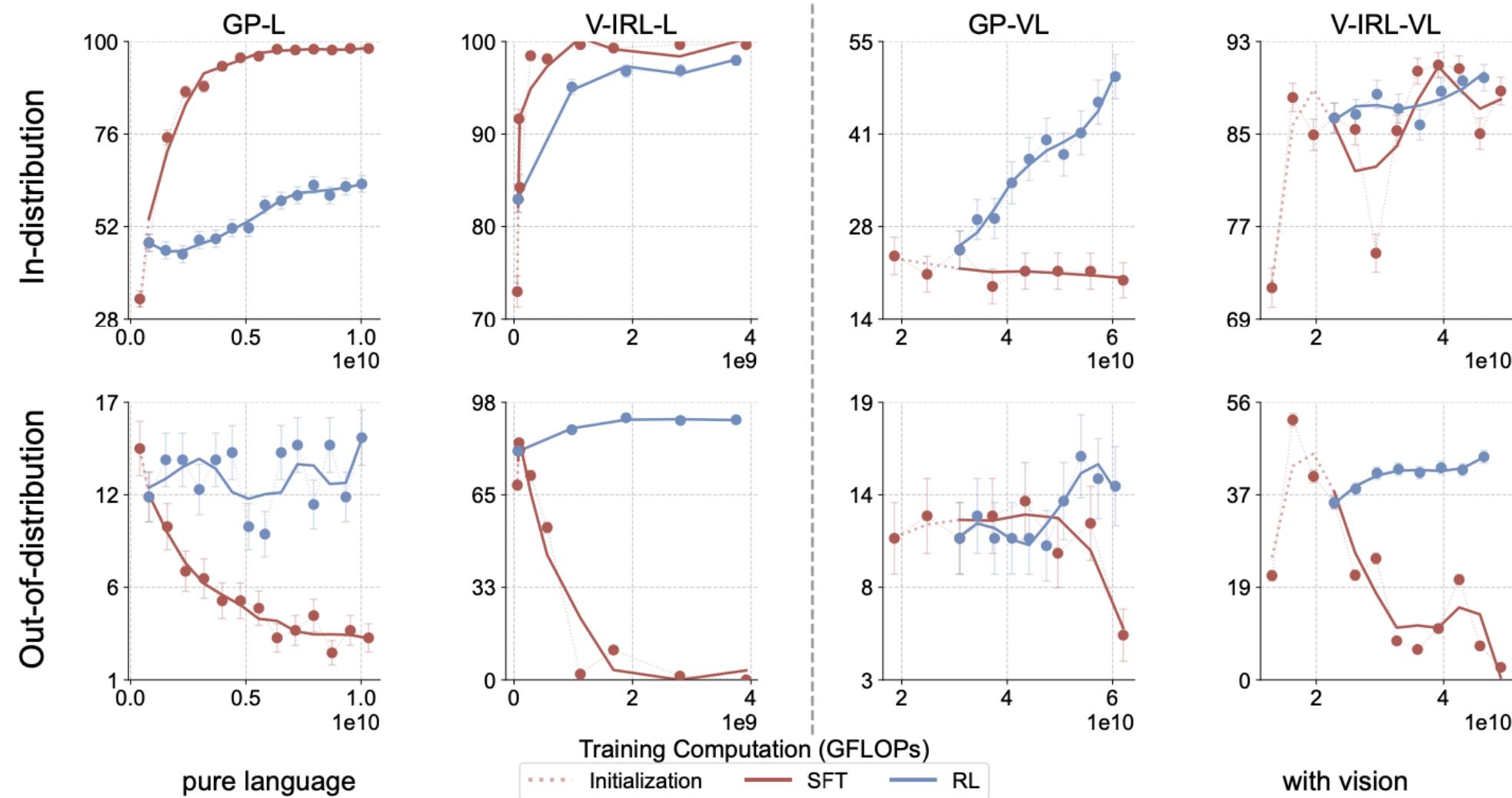
You failed this trial because your formula is incorrect.

$$\triangleright v_{t+1}^{\text{in}} = \text{concat}(v_t^{\text{in}}, v_t^{\text{out}}, v_t^{\text{ver}})$$

Evaluation Task: General Points

- Rule Variations
 - Interpret face cards as all 10 or [11, 12, 13]
 - **[VISUAL]** Use different colored cards (Only Black, Only Red, All)
 - Post Train with one rule, evaluate with the other
- Reward Functions
 - Success: $r = 5$
 - Using each card, but not getting target OR exceeding max verification step of 5: $r = -1$
 - VLM Incorrect recognition: $r = -1.5$
 - Equation not using legal choices: $r = -2$
 - Anything else: $r = -3$

Results: SFT Memorizes, RL Generalizes

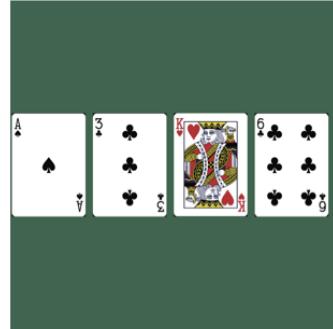


Results: SFT Memorizes, RL Generalizes: SFT Failure Example

System Prompt (v_0^{in})

[Task Description]

You are an expert 24 points card game player. You are observing these four cards in the image. Note that 'J', 'Q', and 'K' count as '10', and each card must be used once. Your goal is to output a formula that evaluates to 24 using numbers from the cards and operators such as '+', '-', '*', '/', '(', ')', and '='.



[Output]

Your response should be a valid json file in the following format:

```
{  
  "cards": [x, y, z, w], where 'J', 'Q', and 'K' count as '10',  
  "number": [a, b, c, d], where a, b, c, and d are the numbers on the cards,  
  "formula": "an equation that equals 24",  
}
```

Appending model and verifier outputs to obtain v_t^{in}

$$v_t^{\text{in}} = [v_0^{\text{out}}, v_0^{\text{ver}}, v_1^{\text{out}}, v_1^{\text{ver}}, \dots, v_{t-1}^{\text{out}}, v_{t-1}^{\text{ver}}]$$

$$\triangleright v_t^{\text{in}} = \text{concat}(v_0^{\text{in}}, [v_k^{\text{out}}, v_k^{\text{ver}}]_{k=0}^{t-1})$$

Model output (v_t^{out})

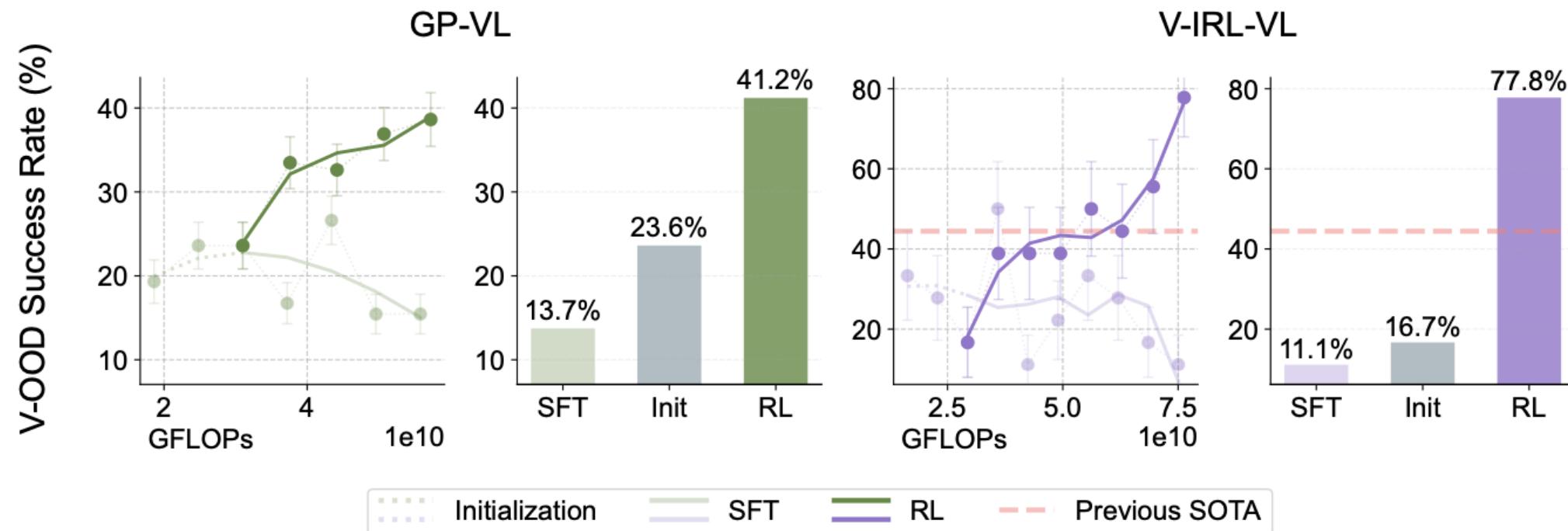
```
{  
  "cards": ['A', '3', 'K', '6'],  
  "number": [1, 3, 13, 6],  
  "formula": "(1+6)*3+13=24",  
}
```

Verifier Output (v_t^{ver})

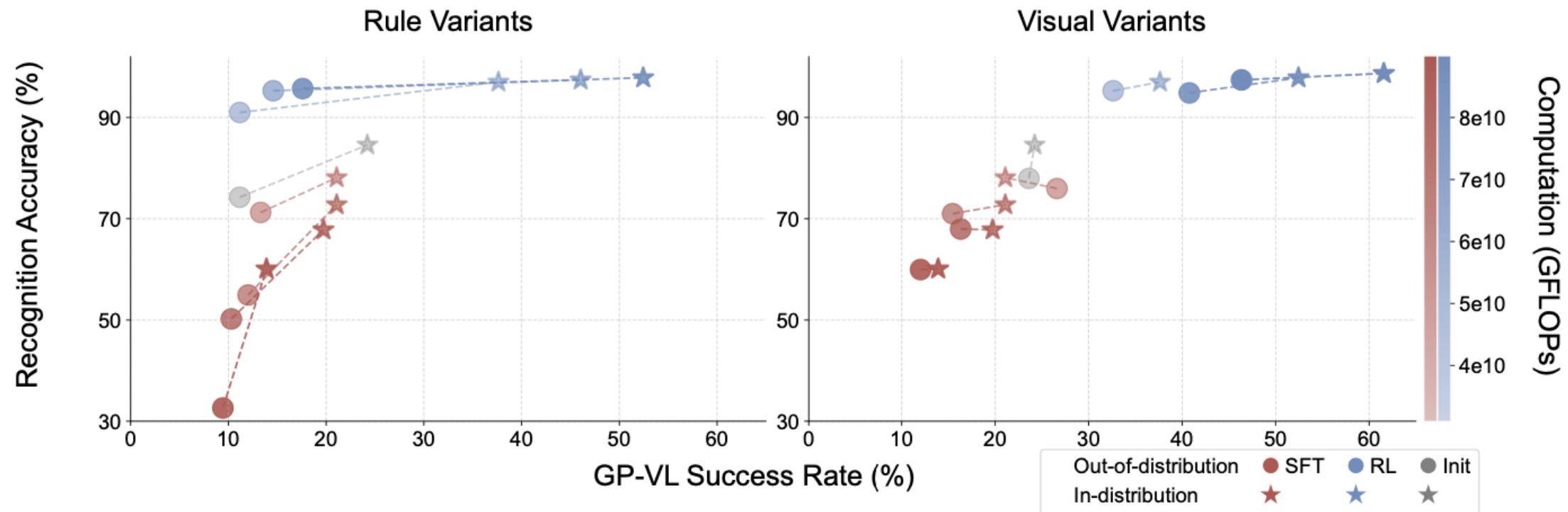
You failed this trial because your formula is incorrect.

$$\triangleright v_{t+1}^{\text{in}} = \text{concat}(v_t^{\text{in}}, v_t^{\text{out}}, v_t^{\text{ver}})$$

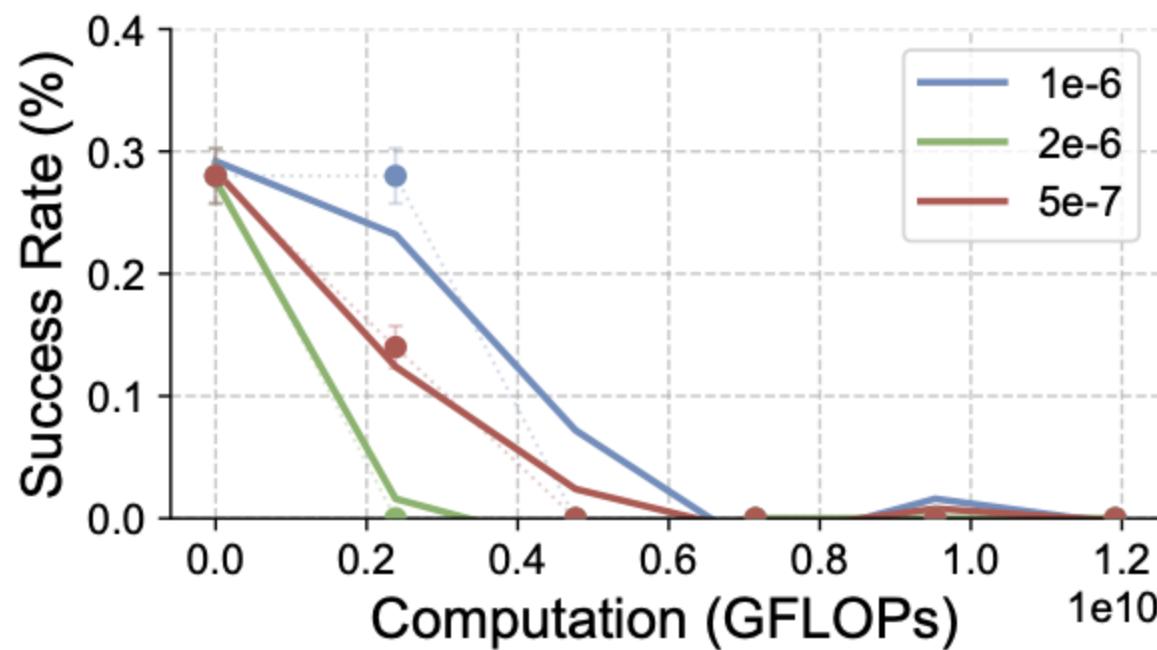
Results: SFT Memorizes, RL Generalizes



Results: Improving VLM Capabilities



Results: RL without SFT



Results: RL without SFT Example

System Prompt (v_0^{in})

[Task Description]

You are an expert 24 points card game player. You are observing these four cards in the image. Note that 'J', 'Q', and 'K' count as '11', '12', and '13' respectively, and each card must be used once. Your goal is to output a formula that evaluates to 24 using numbers from the cards and operators such as '+', '-', '*', '/', '(', ')', and '='.

[Input]

Cards: [2, 'J', '3', 'K']

[Output]

Your response should be a valid json file in the following format:

```
{  
    "cards": [x, y, z, w], where 'J', 'Q', and 'K' count as '10',  
    "number": [a, b, c, d], where a, b, c, and d are the numbers on the cards,  
    "formula": "an equation that equals 24",  
}
```

Appending model and verifier outputs to obtain v_t^{in}

$$v_t^{\text{in}} = [v_0^{\text{out}}, v_0^{\text{ver}}, v_1^{\text{out}}, v_1^{\text{ver}}, \dots, v_{t-1}^{\text{out}}, v_{t-1}^{\text{ver}}]$$

$$\triangleright v_t^{\text{in}} = \text{concat}(v_0^{\text{in}}, [v_k^{\text{out}}, v_k^{\text{ver}}]_{k=0}^{t-1})$$

Failed model output (v_t^{out})

To solve this problem, we can use a brute force approach by generating all possible combinations of the cards and checking if any of them can be used to form an equation that equals 24.

Here is a Python solution for this problem:

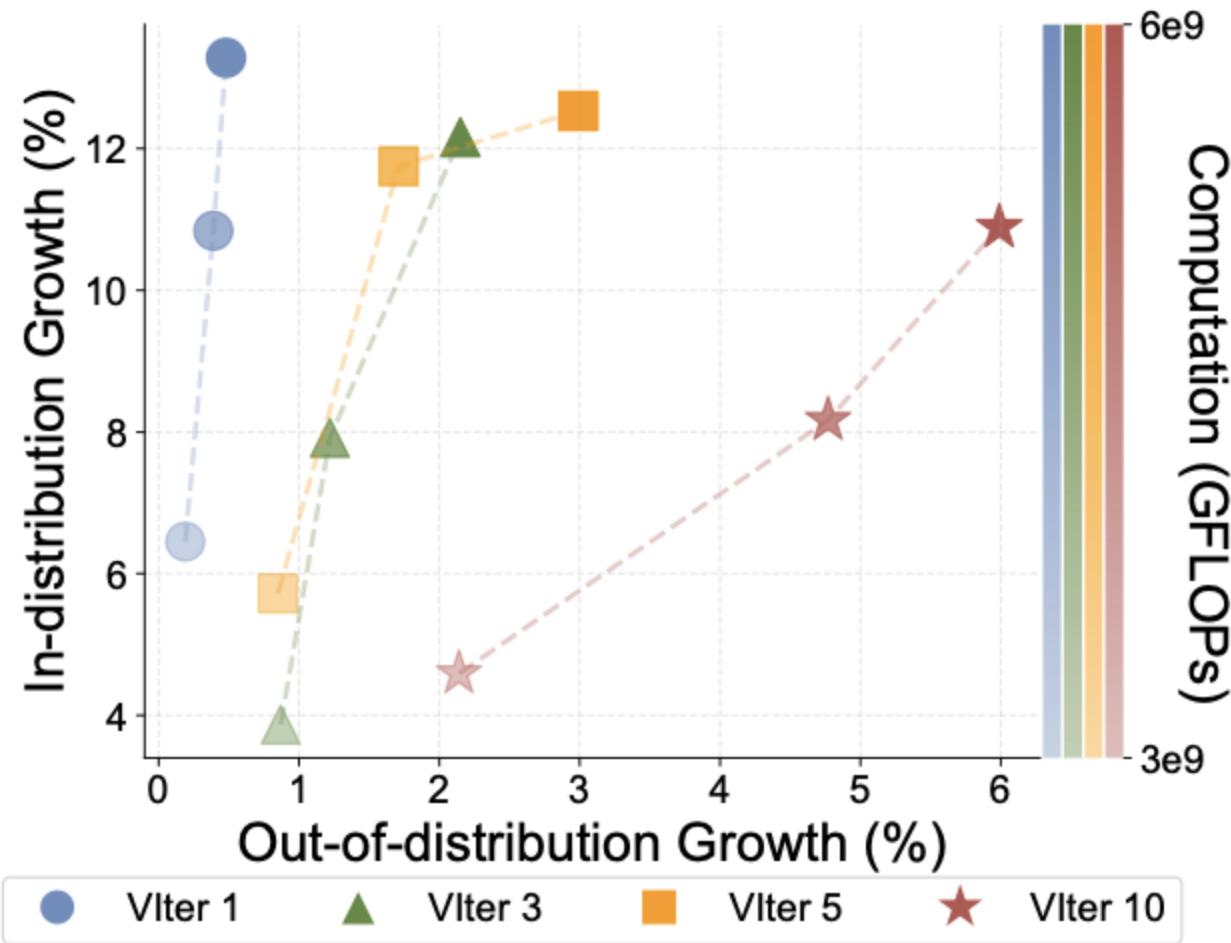
```
““python  
def calculate_formula(cards):  
# Define the mapping of cards to their corresponding numbers  
...““
```

Verifier Output (v_t^{ver})

You failed this trial because your formula is incorrect.

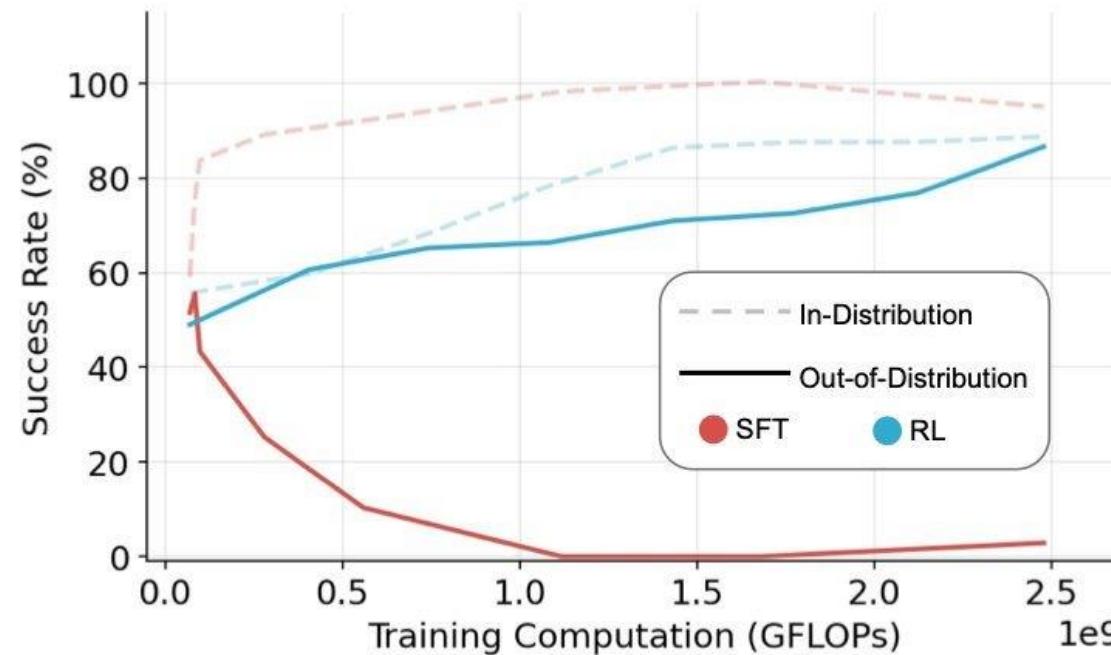
$$\triangleright v_{t+1}^{\text{in}} = \text{concat}(v_t^{\text{in}}, v_t^{\text{out}}, v_t^{\text{ver}})$$

Results: Verification Experimentation



Key Takeaways

- Standardized Fine Tuning Memorizes, Reinforcement Learning Generalizes!
 - Still need SFT for RL initialization. Need to teach proper formatting!



Appendix

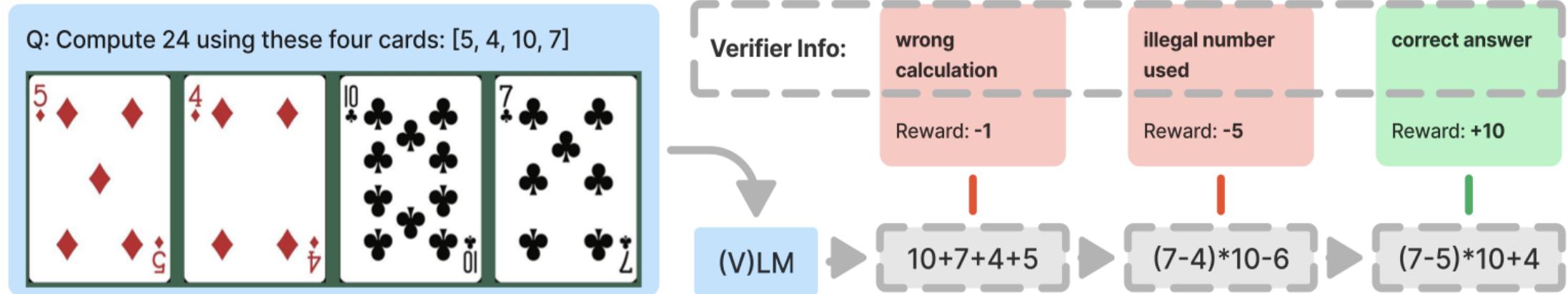


Figure 2: An example of the sequential revision formulation with a verifier. The model generate the next answer v_{t+1}^{out} conditioned on *all previous answers and information* ($v_i^{\text{out}}, v_t^{\text{ver}}, 0 \leq i \leq t$) from the verifier.

System Prompt (v_0^{in})

[Task Description] You are an expert in {task name}, you are observing {purely language/vision-language inputs + <image>}. You are currently at {state related info}. Please follow {tasks rules}.

[Output] Your response should be a valid json file in the following format:
{task related information and answer}

Appending previous model and verifier outputs to obtain v_t^{in}

$$v_t^{\text{in}} = [v_0^{\text{out}}, v_0^{\text{ver}}, v_1^{\text{out}}, v_1^{\text{ver}}, \dots, v_{t-1}^{\text{out}}, v_{t-1}^{\text{ver}}] \quad \triangleright v_t^{\text{in}} = \text{concat}(v_0^{\text{in}}, [v_k^{\text{out}}, v_k^{\text{ver}}]_{k=0}^{t-1})$$

Model output (v_t^{out}) and Verifier Output (v_t^{ver})

$$\{\text{Task related json outputs}\}, \{\text{You success/fail}\}. \quad \triangleright v_{t+1}^{\text{in}} = \text{concat}(v_t^{\text{in}}, v_t^{\text{out}}, v_t^{\text{ver}})$$

Figure 3: An template of our prompt update for constructing v_{t+1}^{in} . The brown parts marks the task and related information, and the purple parts denote the state (s_t) specific info. The blue and red describe the output from the model and verifier, respectively.



VLM
Addition ->

System Prompt (v_0^{in})

[Task Description]

You are an expert in navigation. You will receive a sequence of instructions to follow while observing your surrounding street views. You are also provided with your observation and action history in text. Your goal is to take the action based on the current observation and instruction.

[Instruction]

1. First, turn left to face east.
2. Move forward until you reach the next intersection where Hotel 32One is on your right behind.
3. Turn right to face north.
4. Move forward until you reach the next intersection where Dragon Gate Chinatown SF is on your right front.
5. Turn left to face east.
6. Move forward until the destination Café de la Presse is on your right.



[Current observation]

You observe a 2x2 grid of street view images with the following headings:

[front, right
back, left]

You need to identify if any of the landmarks in the instruction are visible in the street view grid.

[Action space]

- "forward()": indicates moving forward for 1 step;
- "turn_direction(x)": indicates turn direction to the target heading, where $x \in [\text{'north'}, \text{'northeast'}, \text{'east'}, \text{'southeast'}, \text{'south'}, \text{'southwest'}, \text{'west'}, \text{'northwest'}]$;
- "stop()": indicates the navigation is finished;

[Observations and actions sequence]

O_1: No landmarks nearby;

A_1: turn_direction(east)

O_2: No landmarks nearby;

A_2: forward()

O_3: No landmarks nearby;

A_3: forward()

O_4: You observe an image of 4 views; You observe an intersection

A_4:

[Output]

Your response should be a valid json file in the following format:

```
{  
  "current observation": latest observation from the street view grid,  
  "current instruction": analyze the full instruction and identify the sentence to be executed,  
  "action": the action to be taken chosen from the action space,  
}
```

Appending model and verifier outputs to obtain v_t^{in}

$$v_t^{\text{in}} = [v_0^{\text{out}}, v_0^{\text{ver}}, v_1^{\text{out}}, v_1^{\text{ver}}, \dots, v_{t-1}^{\text{out}}, v_{t-1}^{\text{ver}}]$$

$$\triangleright v_t^{\text{in}} = \text{concat}(v_0^{\text{in}}, [v_k^{\text{out}}, v_k^{\text{ver}}]_{k=0}^{t-1})$$

Model output (v_t^{out})

```
{  
  "current observation": "Hotel 32One is on my right behind; I observe an intersection",  
  "current instruction": "Turn right to face north.",  
  "action": "turn_direction(north)",  
}
```

Verifier Output (v_t^{ver})

Correct solution.

O_4: Hotel 32One is on your right behind; You observe an intersection

▷ Move to the next scene upon correct action

- Rule Variations
 - Relative Directions vs. Absolute Direction
 - "Left" vs. East
 - Different Target Locations in Action Space
 - [VISUAL] New York vs. Original Combination
- Reward Functions
 - Success: $r = 1$
 - Wrong Action OR Exceeding verification steps (2): $r = -1$
 - Failed Verification of Landmarks: $r = -1.5$