北京大学
PEKING UNIVERSITY

Memory-Efficient LLM Training via Implicit Structures

Kun Yuan (袁 坤)

MELON Group，Peking University

Jan. 8, 2026

Our Group

北京大学 PEKING UNIVERSITY

Machin**E** **L**earning and **O**ptimizatio**N** (**MELON**) Group at Peking University

MELON

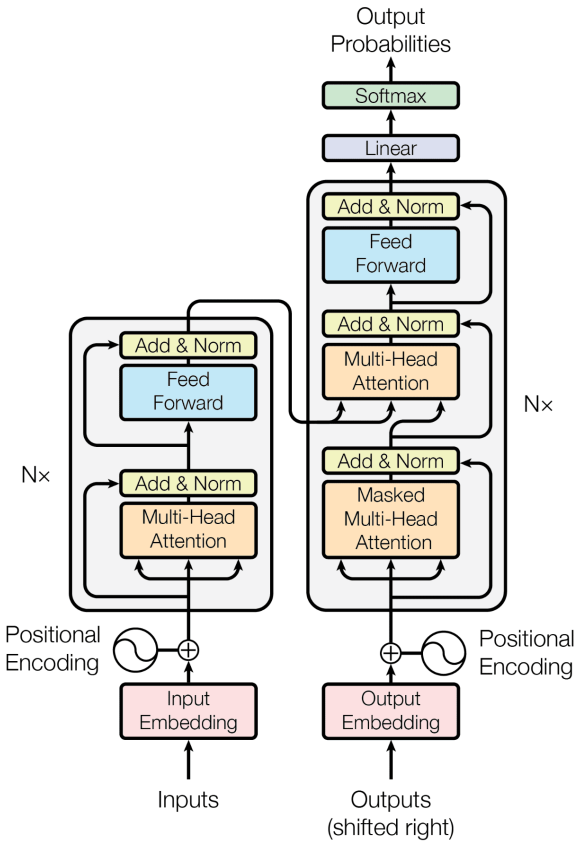
Machine Learning and Optimization

北京大学
PEKING UNIVERSITY

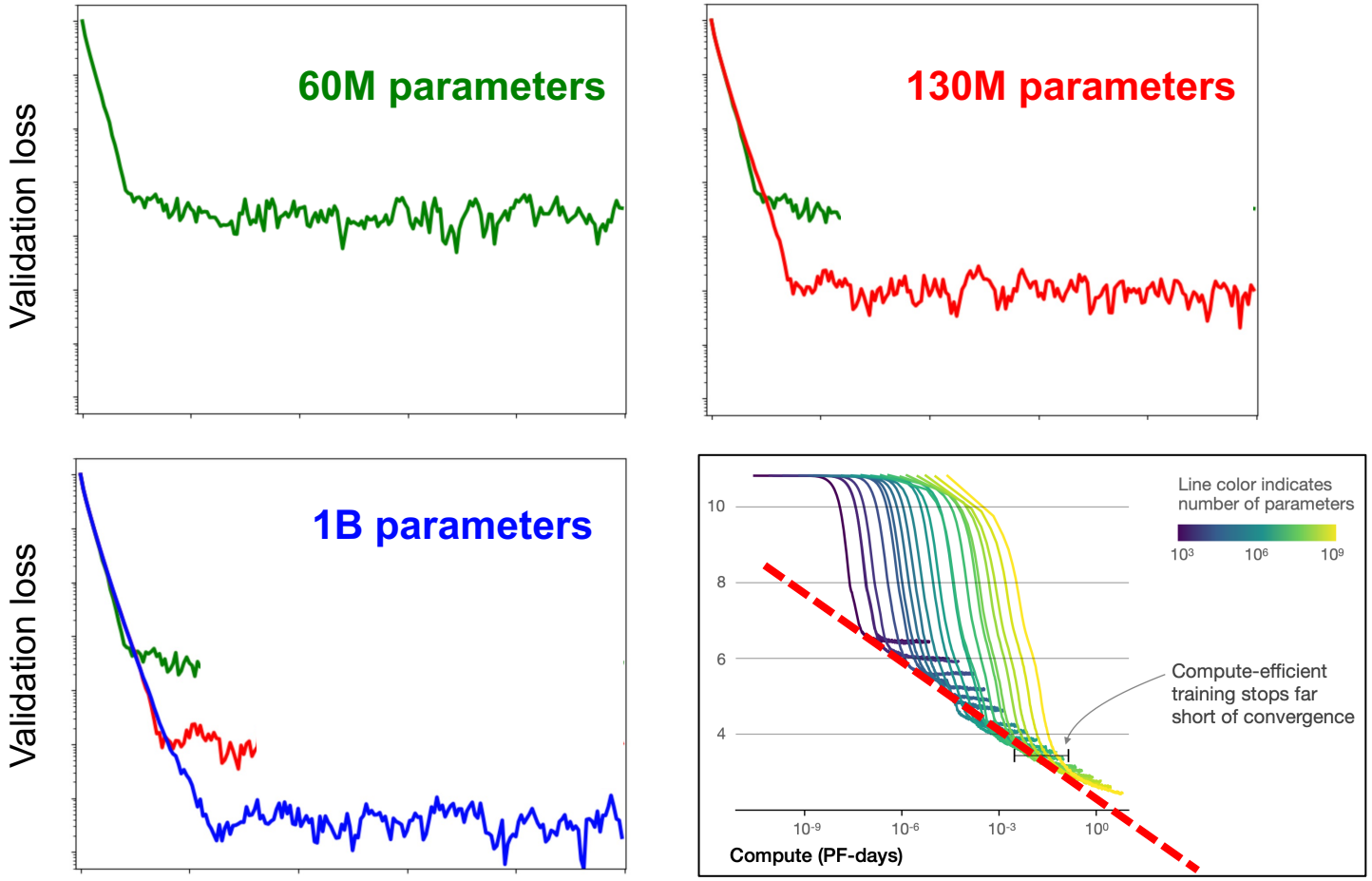
PART 00

Background

Scaling Laws for Large Language Models



LLM Performance vs. Model Size

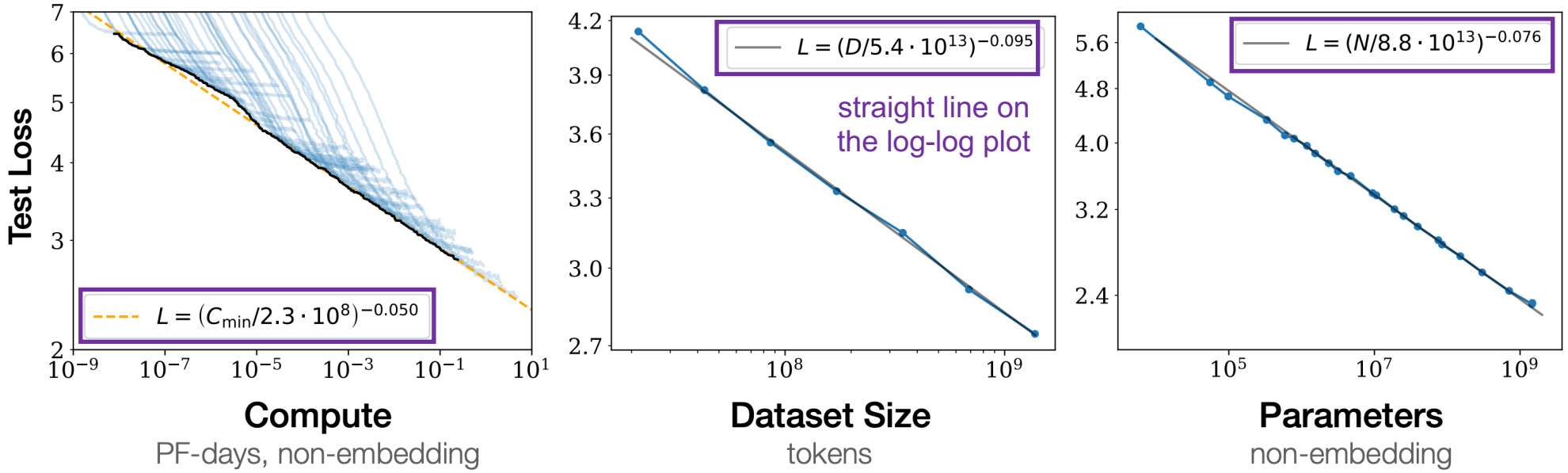


LLM performance improves as parameters gets large

Evaluated across models of different scales

Scaling Laws for Large Language Models

[OpenAI, Scaling Laws for Neural Language Models, 2020]

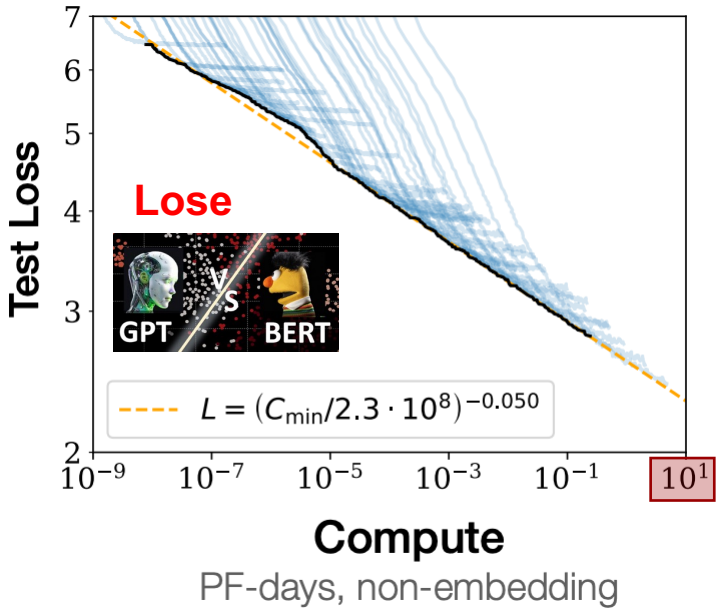


- LLM performance is largely independent of specific architecture
- Performance improves primarily with increases in **data**, **parameters**, and **compute**
- Loss scales as a **power law** with data, parameters, and compute

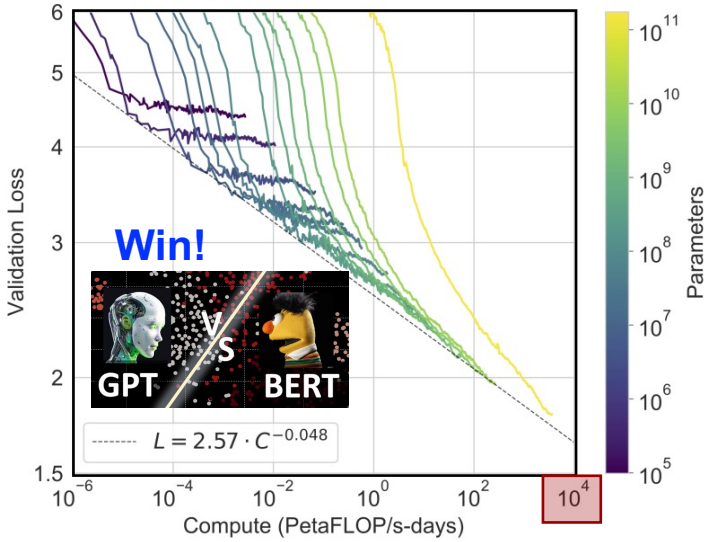
Scaling Laws as a Foundation of ChatGPT's Success

北京大学 PEKING UNIVERSITY

GPT-2, **1.5B** parameters, 2019



GPT-3, **175B** parameters, 2020

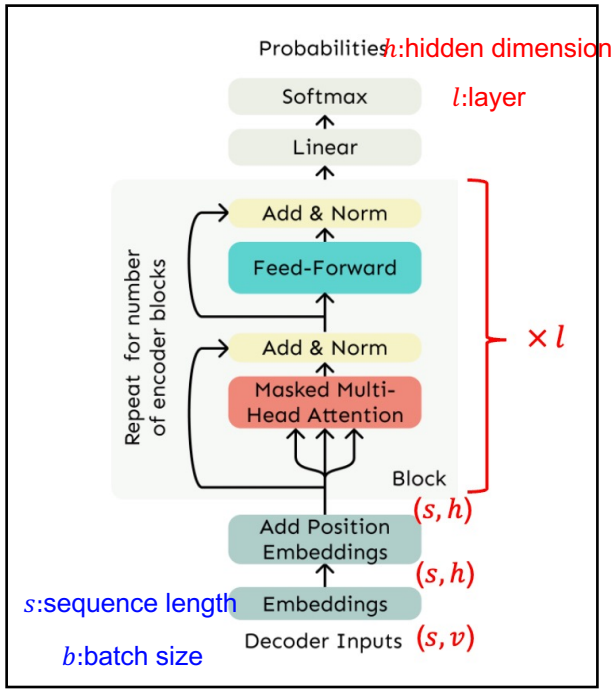


- GPT-2 (2019) scaled far beyond BERT in parameters, yet underperformed in practice
- OpenAI exploited scaling laws to build 175B-param GPT-3, launching the large-model era
- Consensus: **Large model** + **Large data** + **Large compute** = **High intelligence**

Scaling Laws Drive Rapid Growth in Memory Demand for LLMs

- As data size (D) and parameter count (P) increase, memory usage of large models grows rapidly
- In other words, improving large model performance comes at a significant memory cost

Memory = **Model** + **Gradient** + **Optimizer States** + **Activations**

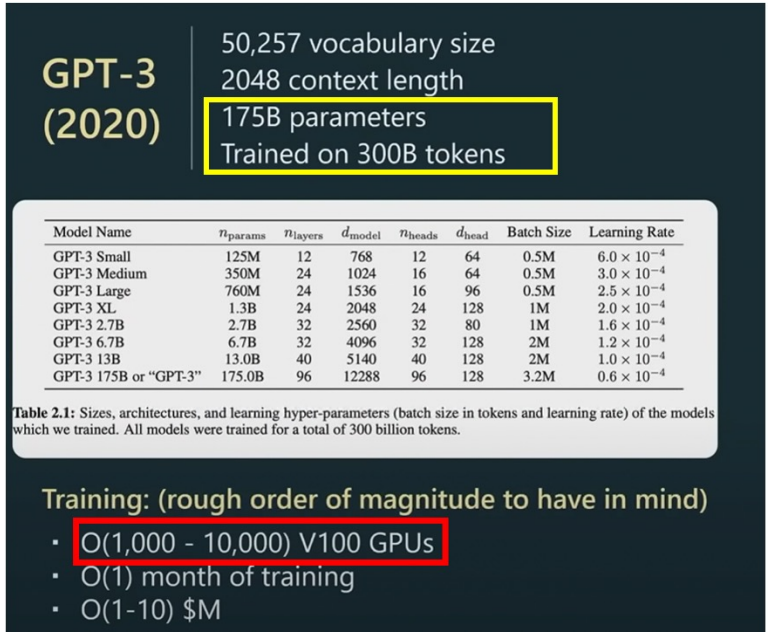


Memory for model, gradients, and optimizer states: $\theta(lh^2) \sim \theta(P)$

- Scales linearly with parameters
- **more parameters, higher memory cost**

Memory for activations: $\Omega(bslh + bls^2a) \sim \Omega(D)$

- Scales at least linearly with data size
- $D = bs$; **more data, higher memory cost**



Scaling Laws Drive Rapid Growth in Memory Demand for LLMs

北京大学 PEKING UNIVERSITY

Training and inference costs rise sharply

↑ Cost grows

Dramatic memory cost; Massive clusters for distributed training

↑ Memory grows

Increasing parameters and data

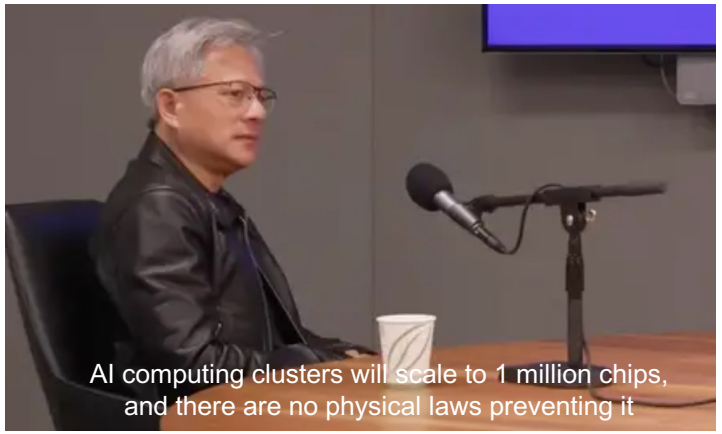
↑ Scaling law

Growing demand for better LLM performance

LLaMA-3 on 20,000 GPUs cluster

To train our largest Llama 3 models, we combined three types of parallelization: data parallelization, model parallelization, and pipeline parallelization. Our most efficient implementation achieves a compute utilization of over 400 TFLOPS per GPU when trained on 16K GPUs simultaneously. We performed training runs on two custom-built 24K GPU clusters. To maximize GPU uptime, we developed an advanced new training stack that automates error detection, handling, and maintenance. We also greatly improved our

XAI built cluster with 100K GPUs



Jensen Huang: AI compute clusters will scale up to 1M chips

Key question: How to save memory?

Scaling Laws Drive Rapid Growth in Memory Demand for LLMs

Training and inference costs rise sharply

Cost scaling

Dramatic memory cost; Massive clusters for distributed training

Memory scaling

Increasing parameters and data

Power law

Growing demand for better LLM performance

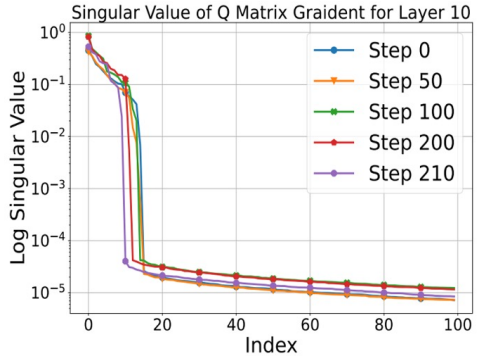
Approach 1: Design new architectures and explore novel scaling laws

Approach 2: Develop new hardware (e.g., SuperNodes) to reduce training and inference costs

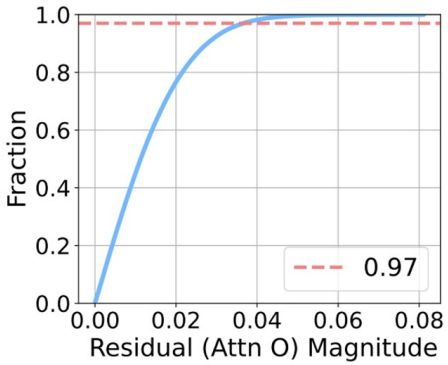
Approach 3: Memory-efficient training methods driven by implicit structures inside models

Insight: As scale increases, LLMs contain significant **structural redundancy**

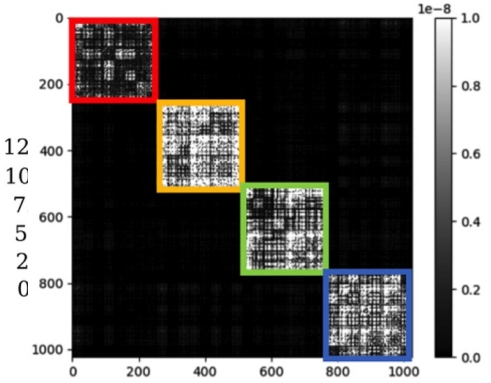
Conventional training wastes memory by ignoring implicit structure



Low-Rank Gradients in LLMs
(low-rank subspace compute & memory))



Sparse FFN Activations
(sparse masking in compute & memory)



Block-Diagonal Hessian Structure
(local/global decoupled compute & memory)

Overview

北京大学 PEKING UNIVERSITY

Goal: **Exploit implicit structures for memory-efficient LLM training**

❶	Low-rank gradients	Develop subspace projection training based on low-rank properties save optimizer states (dense/MoE models) (He-Yuan, ICML 2025; Chen-Yuan, ICML 2025;Chen-Yuan, ICLR 2025)	**Optimizer States**
❷	Sparse FFN layers	Develop importance sampling–based training based on FFN sparsity save activations (dense/MoE models) (Song-Yuan, ICML 2025; Zhu-Yuan, NeurIPS 2024;He-Yuan, ICML 2024;)	**Activations**
❸	Cross-layer low-rank activations	Develop parameter-efficient training based on inter-layer low-rank structure save model and gradients (dense/MoE models) (Kong-Yuan, 2025; Wu-Yuan, 2025)	**Model and Gradients**

北京大学
PEKING UNIVERSITY

PART 01

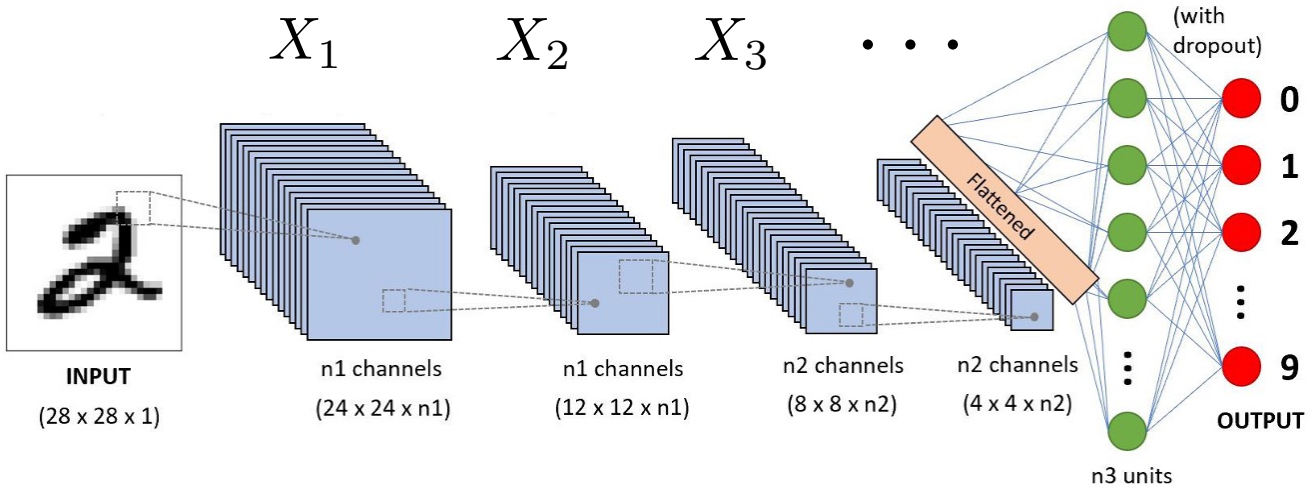
Save Optimizer States via Low-Rank Gradient

Y. He, P. Li, Y. Hu, C. Chen, K. Yuan, *Subspace Optimization for Large Language Models with Convergence Guarantees*, ICML 2025.

Y. Chen, Y. Zhang, Y. Liu, K. Yuan, Z. Wen, *A Memory Efficient Randomized Subspace Optimization Method for Training Large Language Models*, ICML 2025

LLM pretraining is essentially solving stochastic optimization

- The model weights in neural networks are a set of matrices $\boldsymbol{X} = \{X_\ell\}_{\ell=1}^{L}$



- Let $h(\boldsymbol{X};\xi)$ be the language model; $\hat{y} = h(\boldsymbol{X};\xi)$ is the predicted token

cross entropy

LLM cost function: $$\boldsymbol{X}^\star = \arg\min_{\boldsymbol{X}} \left\{ \mathbb{E}_{\xi\sim\mathcal{D}} \left[L(h(\boldsymbol{X};\xi), y) \right] \right\}$$

data distribution pred. token real token

LLM pretraining is essentially solving stochastic optimization

- If we define $\boldsymbol{\xi} = (\xi, y)$ and $F(\boldsymbol{X}; \boldsymbol{\xi}) = L(h(\boldsymbol{X}; \xi), y)$, the LLM problem becomes

$$\text{Stochastic optimization:} \quad \boldsymbol{X}^\star = \arg\min_{\boldsymbol{X}} \left\{ \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{D}} \left[F(\boldsymbol{X}; \boldsymbol{\xi}) \right] \right\}$$

- In other words, LLM pretraining is essentially solving a stochastic optimization problem

- Adam is the standard approach in LLM pretraining

Optimizer states

$$\boldsymbol{G}_t = \nabla F(\boldsymbol{X}_t; \boldsymbol{\xi}_t) \qquad \text{(stochastic gradient)}$$

$$\boldsymbol{M}_t = (1-\beta_1)\boldsymbol{M}_{t-1} + \beta_1 \boldsymbol{G}_t \qquad \text{(first-order momentum)}$$

$$\boldsymbol{V}_t = (1-\beta_2)\boldsymbol{V}_{t-1} + \beta_2 \boldsymbol{G}_t \odot \boldsymbol{G}_t \qquad \text{(second-order momentum)}$$

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t - \frac{\gamma}{\sqrt{\boldsymbol{V}_t} + \epsilon} \odot \boldsymbol{M}_t \qquad \text{(adaptive SGD)}$$

Memory cost to pre-train LLMs

北京大学 PEKING UNIVERSITY

Memory = **Model** + **Gradient** + **Optimizer states** + **Activations**

- Given a **model** with **P** parameters, **gradient** will consume **P** parameters, and **optimizer states** will consume **2P** parameters; **4P parameters in total.**

P $\boldsymbol{G}_t = \nabla F(\boldsymbol{X}_t; \boldsymbol{\xi}_t)$

2P
$$\boldsymbol{M}_t = (1-\beta_1)\boldsymbol{M}_{t-1} + \beta_1 \boldsymbol{G}_t$$
$$\boldsymbol{V}_t = (1-\beta_2)\boldsymbol{V}_{t-1} + \beta_2 \boldsymbol{G}_t \odot \boldsymbol{G}_t$$

P $$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t - \frac{\gamma}{\sqrt{\boldsymbol{V}_t} + \epsilon} \odot \boldsymbol{M}_t$$

Optimizer states contribute significantly to memory usage

Memory cost to pre-train LLMs

北京大学 PEKING UNIVERSITY

Memory = **Model** + **Gradient** + **Optimizer states** + **Activations**

- Activations are auxiliary variables to facilitate the gradient calculations

Consider a linear neural network

$$z_i = X_i z_{i-1}, \forall\, i = 1, \cdots, L$$
$$f = \mathcal{L}(z_i; y)$$

The gradient is derived as follows

$$\frac{\partial f}{\partial X_i} = \frac{\partial f}{\partial z_i} z_{i-1}^\top$$

Need to store activations $z_1, z_2, \cdots, z_L$

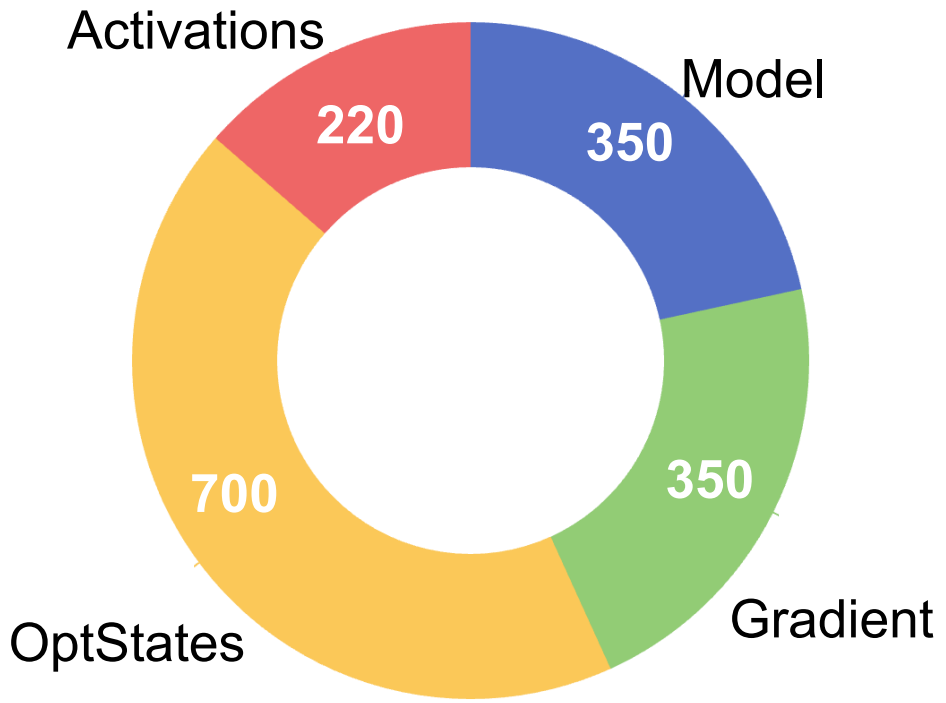
- The size of activations depends on sequence length and batch size

Minimum memory requirement: GPT-3

北京大学 PEKING UNIVERSITY

- Pretrain GPT-3 model (BF16) from scratch with **a single batch size** requires

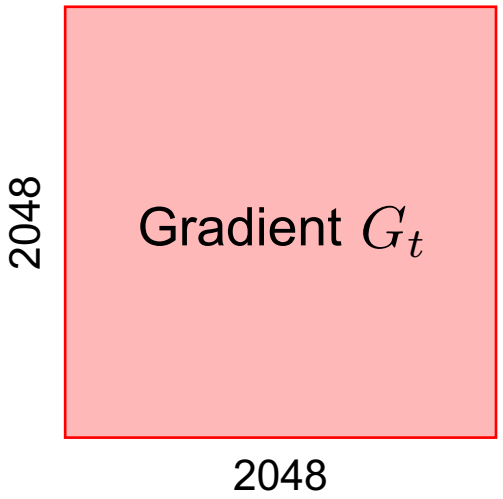
  - Parameters: 175B
  - Model storage: 175B * 2 Bytes = 350 GB
  - Gradient storage: 350 GB
  - Optimizer states: 700 GB (using Adam)
  - Activation storage: ~220 GB
  - In total: **1620 GB**

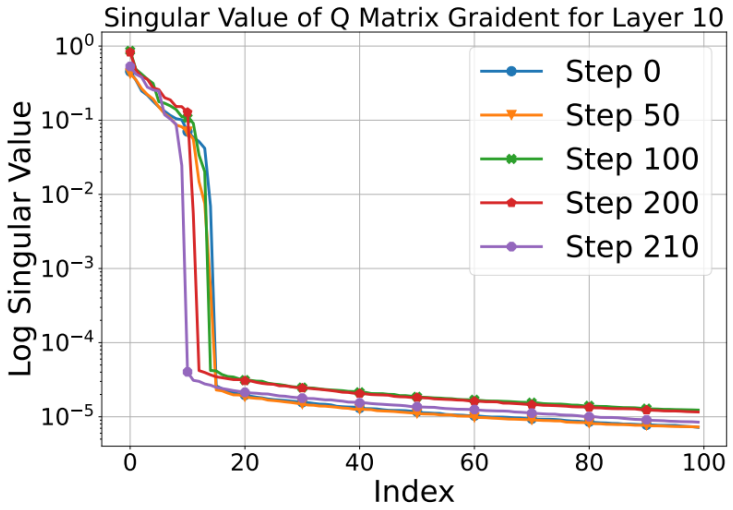


How to save optimizer states?

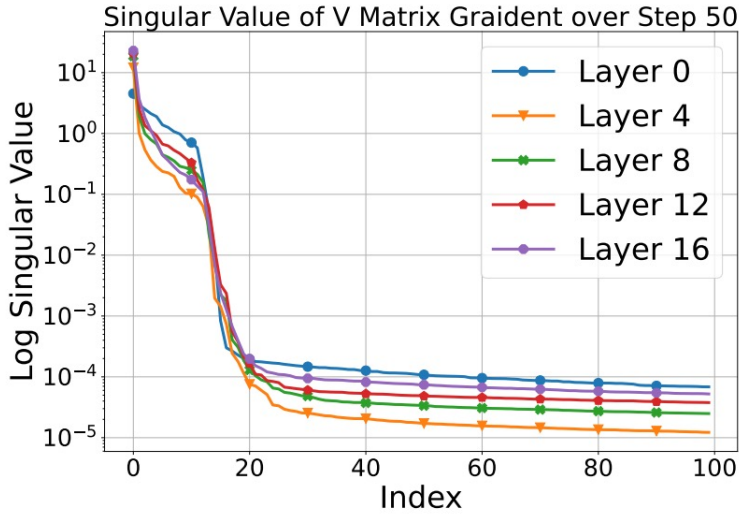
GaLore: Gradient Low-Rank Projection

- Observation: gradient in LLMs becomes **low-rank** during training [1]





remain low-rank across iterations



remain low-rank across layers

- Given a gradient matrix with dimensions 2048 by 2048, around **top 10** eigenvalues dominate

- How to utilize the low-rank structure in gradients?

[1] Y. Chen, et. al., *Enhancing Zeroth-Order Fine-tuning for Language Models with Low-Rank Structures,* ICLR 2025

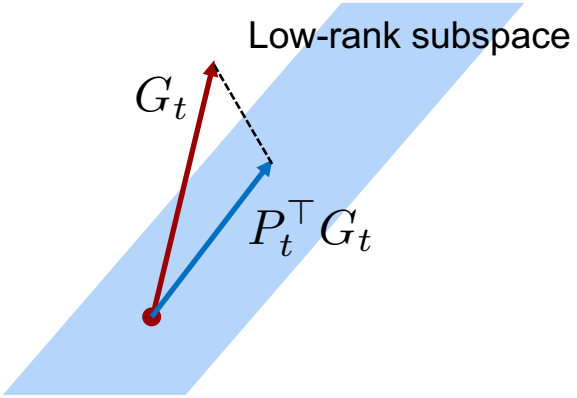
GaLore: Gradient Low-Rank Projection Algorithm

- **Main idea:** Projecting gradient onto the low-rank subspace [1]

- Given gradient $G_t \in \mathbb{R}^{m \times n}$ and projection matrix $P_t \in \mathbb{R}^{m \times r}$, we project high-rank gradient into low-rank subspace:

$$G_t \in \mathbb{R}^{m \times n} \quad \boxed{P_t^\top G_t} \quad g_t = P_t^\top G_t \in \mathbb{R}^{r \times n}$$

Low-rank projection



- When subspace rank $r \ll m$, low-rank gradient g_t has much fewer parameters than G_t

- In practice, m is $10^3 \sim 10^4$, r is $10^1 \sim 10^2$; when gradient G_t has a low rank, g_t is a good gradient estimator

[1] J. Zhao, et. al., *Galore: Memory-efficient LLM training by gradient low-rank projection,* ICML 2024

GaLore: Gradient Low-Rank Projection

- Low-rank optimizer states:

$$
\begin{aligned}
\boldsymbol{g}_t &= \boldsymbol{P}_t^\top \boldsymbol{G}_t && \triangleright \text{ dims r x n} \\
\boldsymbol{m}_t &= (1-\beta_1)\boldsymbol{m}_{t-1} + \beta_1 \boldsymbol{g}_t && \triangleright \text{ dims r x n} \\
\boldsymbol{v}_t &= (1-\beta_2)\boldsymbol{v}_{t-1} + \beta_2 \boldsymbol{g}_t \odot \boldsymbol{g}_t && \triangleright \text{ dims r x n} \\
\boldsymbol{\delta}_t &= \frac{\gamma}{\sqrt{\boldsymbol{v}_t} + \epsilon} \odot \boldsymbol{m}_t && \triangleright \text{ dims r x n}
\end{aligned}
$$

- Parameter updates:

$$
\boldsymbol{X}_{t+1} = \boldsymbol{X}_t - \boldsymbol{P}_t \boldsymbol{\delta}_t \qquad \triangleright \text{ dims m x n}
$$

Simplified as

$$
\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \boldsymbol{P}_t \boldsymbol{\rho}(\boldsymbol{P}_t^\top \boldsymbol{G}_t)
$$

- Memory cost: Model $\boldsymbol{X}$, Gradient $\boldsymbol{G}$, Projection $\boldsymbol{P}$, OptStates $\boldsymbol{m}, \boldsymbol{v}$ and activations

trivial memory cost

J. Zhao, et. al., *Galore: Memory-efficient LLM training by gradient low-rank projection,* ICML 2024

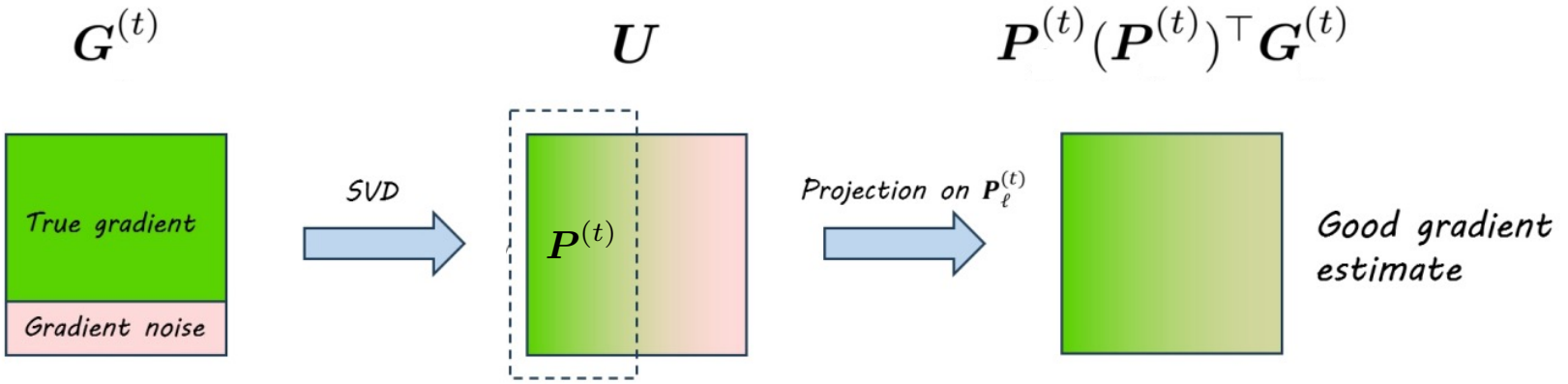
Updating optimizer states via low-rank projected gradients

- Low-rank training: $\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \boldsymbol{P}_t \rho(\boldsymbol{P}_t^\top \boldsymbol{G}_t)$

- How to achieve the low-rank projection matrix? **Singular Value Decomposition!**

$$G_t = U\Sigma V^\top \longrightarrow P_t = \boxed{U[:, :r]} \in \mathbb{R}^{m\times r}$$

Select the dominant top-r columns



Yutong He, Kun Yuan, et. al., *Subspace Optimization for Large Language Models with Convergence Guarantees*, ICML 2025

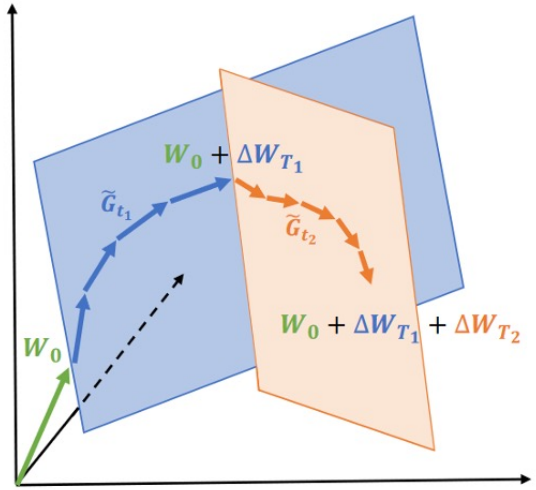
Updating optimizer states via low-rank projected gradients

- It is computationally expensive to perform SVD in each iteration
- **Lazy-SVD**: perform SVD every τ iterations； using the same projector otherwise [1]

Low-rank training algorithm based on lazy-SVD (GaLore)

$$\begin{cases} \boldsymbol{P}_t \leftarrow \mathrm{SVD}(\boldsymbol{G}_t) & \text{if } t \bmod \tau = 0 \\ \boldsymbol{P}_t \leftarrow \boldsymbol{P}_{t-1} & \text{otherwise} \end{cases}$$

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \boldsymbol{P}_t \boldsymbol{\rho}(\boldsymbol{P}_t^\top \boldsymbol{G}_t)$$



- Applying SVD every τ steps reduces the computation cost.

[1] J. Zhao, et. al., *Galore: Memory-efficient LLM training by gradient low-rank projection,* ICML 2024

GaLore: Gradient Low-Rank Projection Algorithm

Pretraining LLaMA on C4 dataset [1]

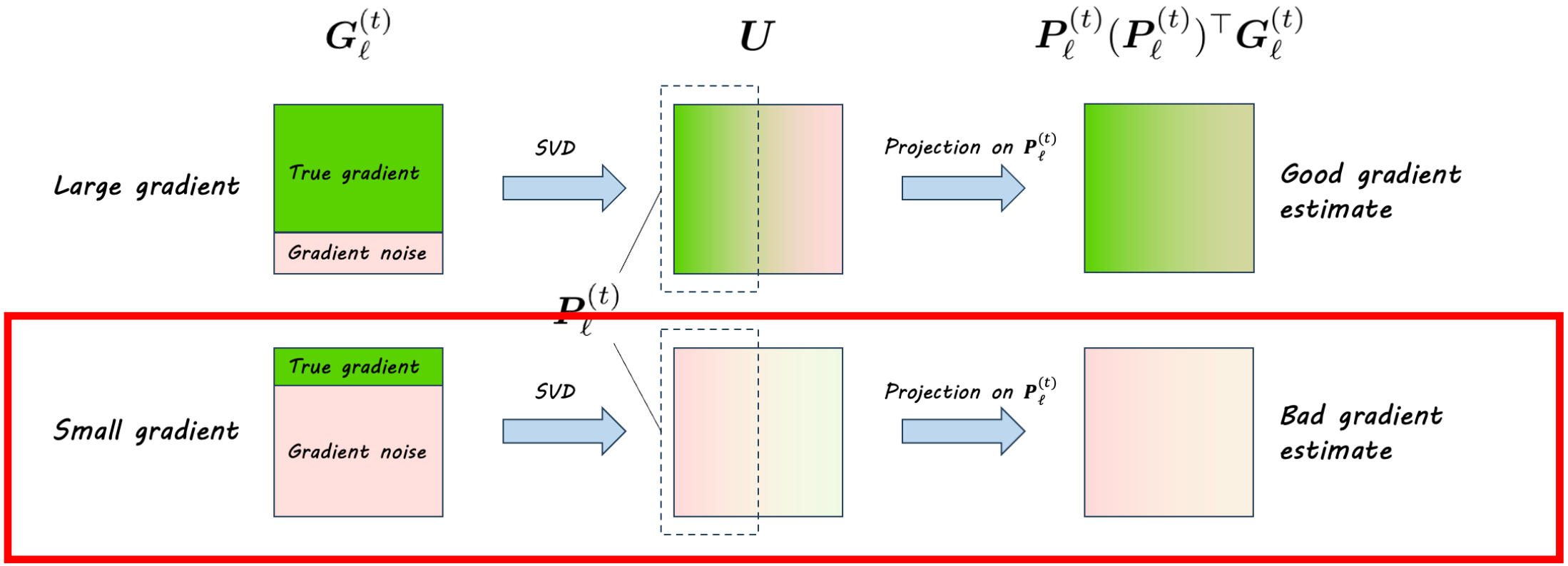
	60M	130M	350M	1B
Full-Rank	34.06 (0.36G)	25.08 (0.76G)	18.80 (2.06G)	15.56 (7.80G)
GaLore	**34.88** (0.24G)	**25.36** (0.52G)	**18.95** (1.22G)	**15.64** (4.38G)
Low-Rank	78.18 (0.26G)	45.51 (0.54G)	37.41 (1.08G)	142.53 (3.57G)
LoRA	34.99 (0.36G)	33.92 (0.80G)	25.58 (1.76G)	19.21 (6.17G)
ReLoRA	37.04 (0.36G)	29.37 (0.80G)	29.08 (1.76G)	18.33 (6.17G)
r/d_{model}	128 / 256	256 / 768	256 / 1024	512 / 2048
Training Tokens	1.1B	2.2B	6.4B	13.1B

[1] J. Zhao, et. al., *Galore: Memory-efficient LLM training by gradient low-rank projection,* ICML 2024

GaLore achieves significant memory saving with under 0.5% performance loss

Does GaLore guarantee convergence to a local minimum or stationary point?

GaLore does not always converge! SVD projection introduces issues

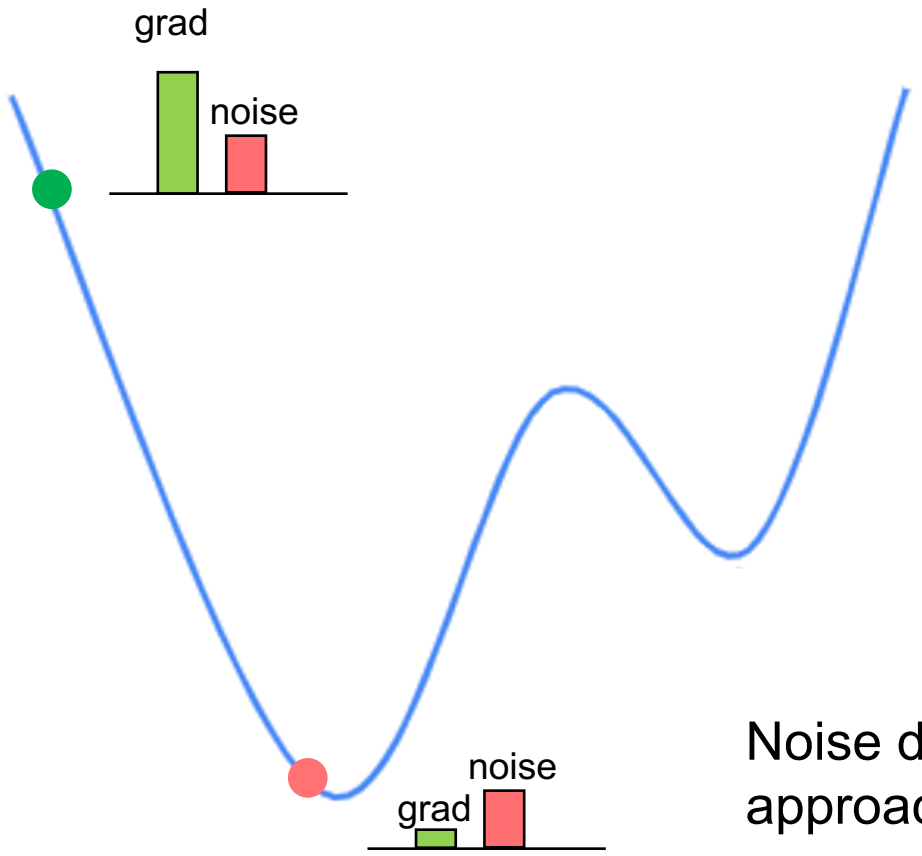


When gradient noise dominates the stochastic gradient, SVD captures **noise-dominated** subspace!

All gradient information is lost !

Is noise-dominance a common case? Yes!

Gradient dominates during the initial stages



Noise dominates when approaching the local minimum

Theoretical flaws of SVD projection: Counterexample construction

北京大学 PEKING UNIVERSITY

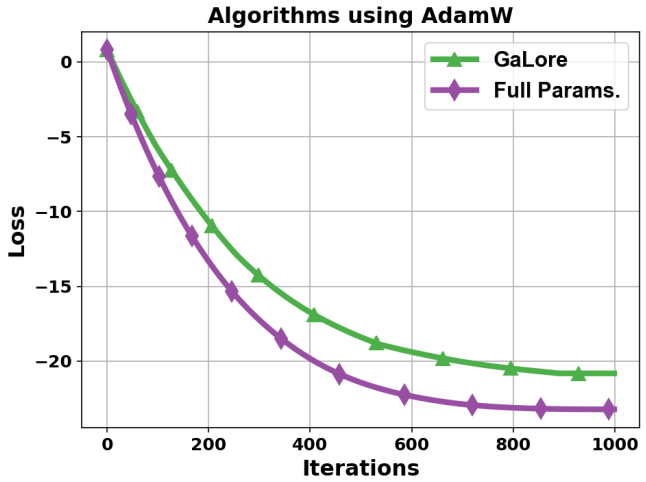
Counter-Example. We consider the following quadratic problem with gradient noise:

$$f(\boldsymbol{X}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{X}\|_F^2 + \langle \boldsymbol{B}, \boldsymbol{X}\rangle_F, \quad \nabla F(\boldsymbol{X};\xi) = \nabla f(\boldsymbol{X}) + \xi\sigma\boldsymbol{C}, \tag{1}$$

where $\boldsymbol{A} = (\boldsymbol{I}_{n-r} \quad 0) \in \mathbb{R}^{(n-r)\times n}$, $\boldsymbol{B} = \begin{pmatrix} \boldsymbol{D} & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{n\times n}$ with $\boldsymbol{D} \in \mathbb{R}^{(n-r)\times(n-r)}$ generated randomly, $\boldsymbol{C} = \begin{pmatrix} 0 & 0 \\ 0 & \boldsymbol{I}_r \end{pmatrix} \in \mathbb{R}^{n\times n}$, ξ is a random variable uniformly sampled from $\{1, -1\}$ per iteration, and σ is used to control the gradient noise.

Theorem (Non-convergence of GaLore): There exists an objective function $f : \mathbb{R}^d \to \mathbb{R}$ satisfying Assumptions 1 and 2, a stochastic gradient oracle (F, D) satisfying Assumption 3, an initial point $\boldsymbol{x}^{(0)} \in \mathbb{R}^d$, and a constant $\epsilon_0 > 0$ such that for any rank $r_\ell < \min\{m_\ell, n_\ell\}$, subspace changing frequency τ, any optimizer ρ that inputs a subspace gradient of shape $r_\ell \times n_\ell$ and outputs a subspace update direction of the same shape, and for any $t > 0$, it holds that

$$\|\nabla f(\boldsymbol{x}^{(t)})\|_2^2 \geq \epsilon_0.$$



GaLore does **NOT** converge to desired solutions

[1] Yutong He, Kun Yuan et al., *Subspace Optimization for Large Language Models with Convergence Guarantees*, ICML 2025

GoLore: Subspace training based on random projection

- **G**radient rand**o**m **Lo**w-rank p**r**oj**e**ction **(GoLore)** randomly projects the gradient via

$$P_t \sim \mathcal{U}(\mathrm{St}_{m,r})$$

Lemma 5 (Error of GoLore's projection). *Let* $\boldsymbol{P} \sim \mathcal{U}(\mathrm{St}_{m,r})$, $\boldsymbol{Q} \sim \mathcal{U}(\mathrm{St}_{n,r})$, *it holds for all* $\boldsymbol{G} \in \mathbb{R}^{m\times n}$ *that*

$$\mathbb{E}[\boldsymbol{P}\boldsymbol{P}^\top] = \frac{r}{m}\cdot \boldsymbol{I}, \quad \mathbb{E}[\boldsymbol{Q}\boldsymbol{Q}^\top] = \frac{r}{n}\cdot \boldsymbol{I},$$

and

$$\mathbb{E}[\|\boldsymbol{P}\boldsymbol{P}^\top \boldsymbol{G} - \boldsymbol{G}\|_F^2] = \left(1-\frac{r}{m}\right)\|\boldsymbol{G}\|_F^2, \quad \mathbb{E}[\|\boldsymbol{G}\boldsymbol{Q}\boldsymbol{Q}^\top - \boldsymbol{G}\|_F^2] = \left(1-\frac{r}{n}\right)\|\boldsymbol{G}\|_F^2.$$

$$\mathbb{E}[PP^\top G] = \mathbb{E}[PP^\top]\cdot\mathbb{E}[G] = \frac{r}{m}\nabla F(X)$$

The low-rank randomized projected gradient is an unbiased estimate of the true gradient

GoLore's theoretical convergence guarantee

北京大学 PEKING UNIVERSITY

Theorem (Convergence rate of GoLore): Under Assumptions 1-3, for any $T \geq 2 + 128/(3\underline{\delta}) + (128\sigma)^2/(9\sqrt{\underline{\delta}}L\Delta)$, GoLore using small-batch stochastic gradients and MSGD with MP converges as

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\|\nabla f(\boldsymbol{x}^{(t)})\|_2^2\right] = \mathcal{O}\left(\frac{L\Delta}{\underline{\delta}^{5/2}T} + \sqrt{\frac{L\Delta\sigma^2}{\underline{\delta}^{7/2}T}}\right),$$

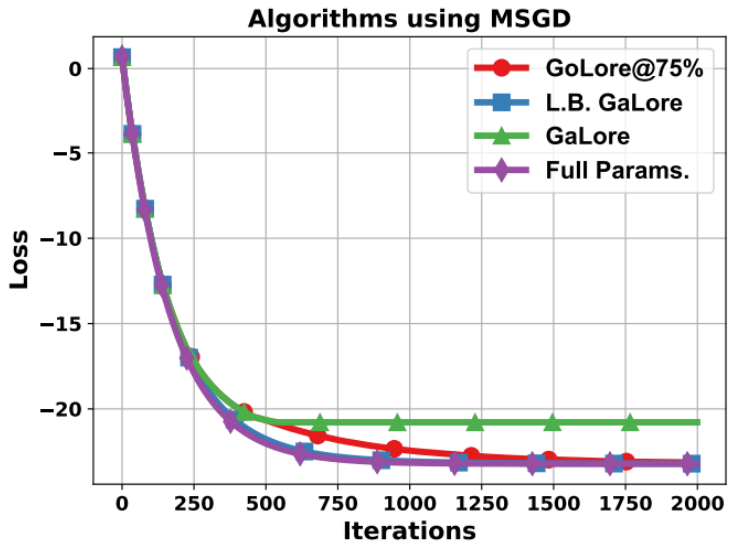
where $\Delta = f(\boldsymbol{x}^{(0)}) - \inf_{\boldsymbol{x}} f(\boldsymbol{x})$ and $\underline{\delta} := \min_\ell \frac{r_\ell}{\min\{m_\ell, n_\ell\}}$.

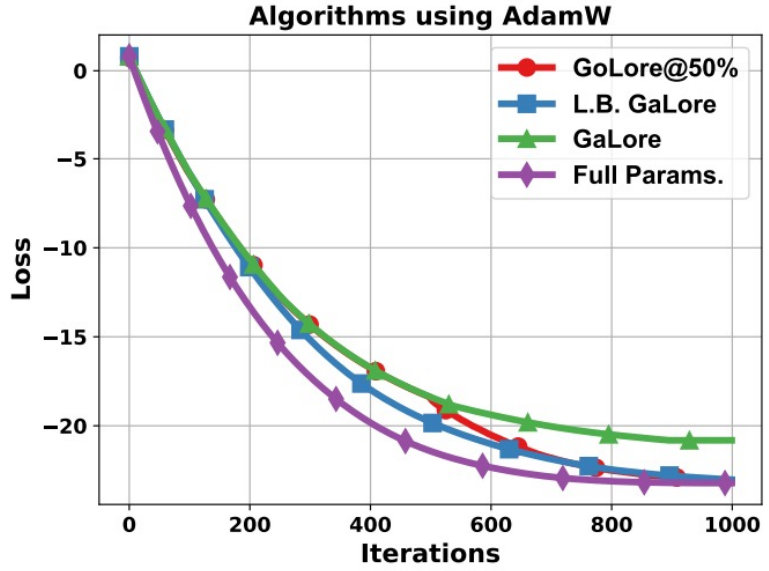
- GoLore is guaranteed to converge at a rate of $\mathcal{O}(1/\sqrt{T})$.
- Adam converges at $\mathcal{O}(1/\sqrt{T})$, implying low-rank projection keeps the convergence order

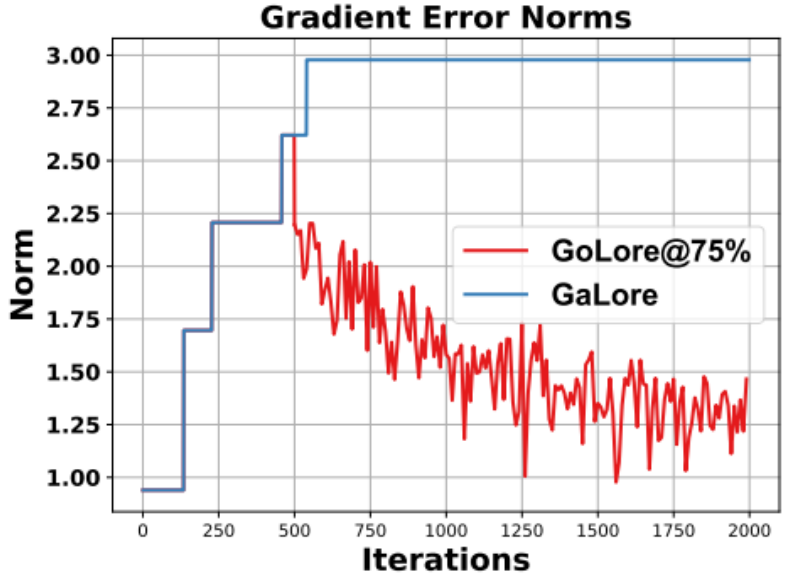
A hybrid strategy: random projection + SVD

- SVD projection is preferred in initial stages: effectively capture gradient information
- Random projection is preferred when approaching solutions: avoid losing gradient information

GoLore@x% = GaLore (first (100-x)% iters) + GoLore (last x% iters)

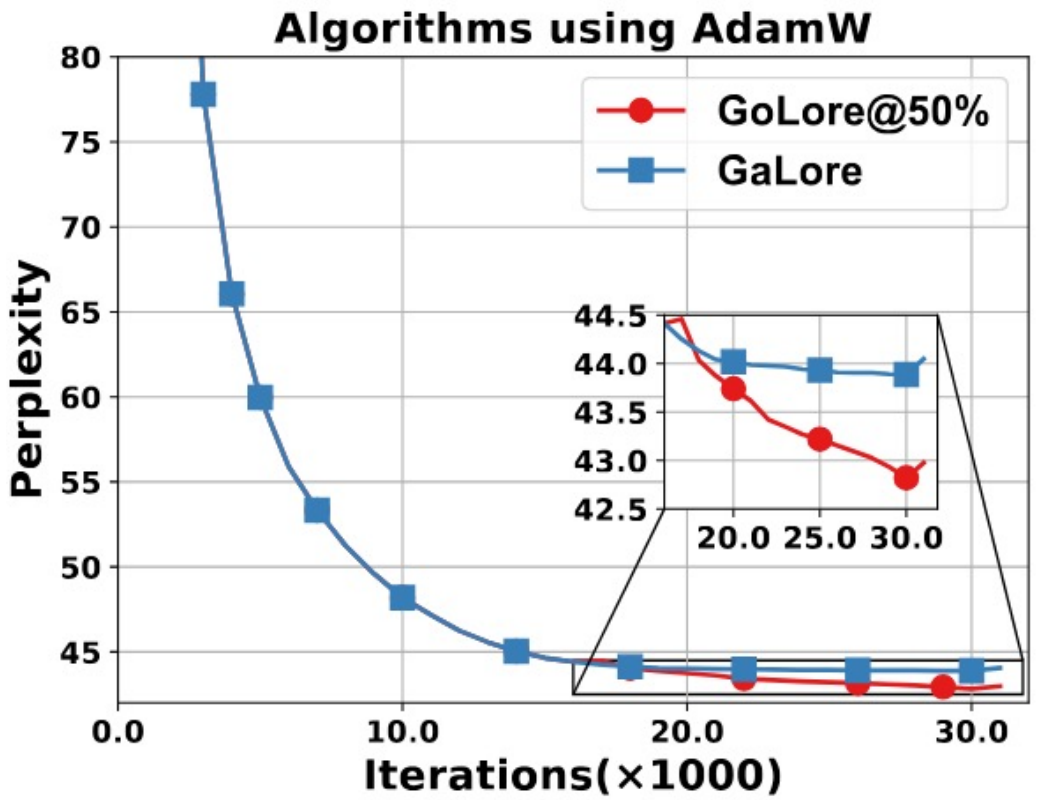




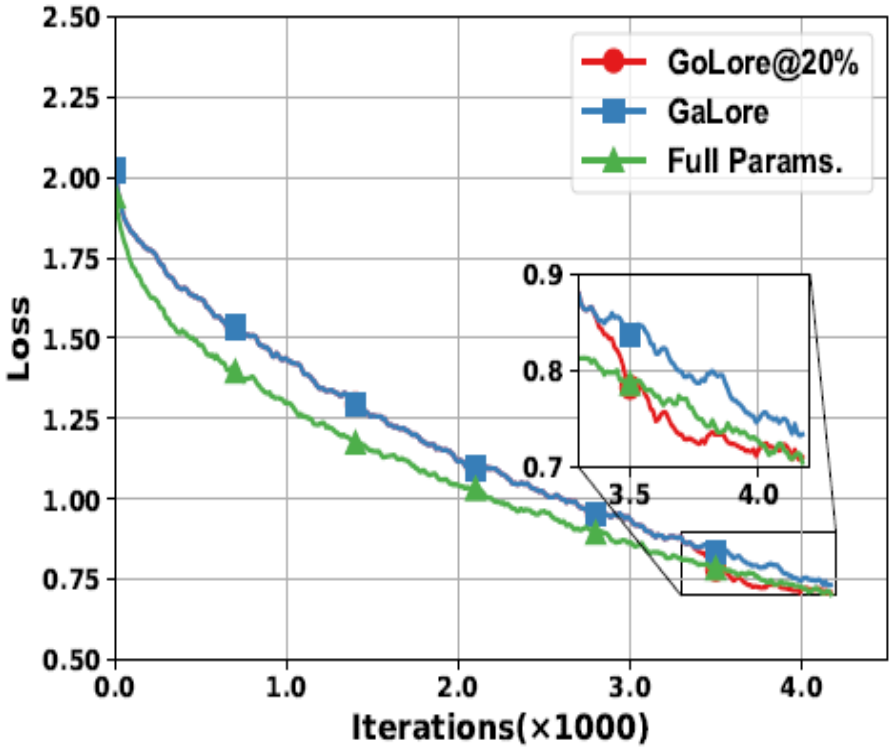


Experimental results on fine-tuning

Pre-train LLaM2-60M on C4



Fine-tuning LLaMA2-7B on WinoGrande:



Experimental results on fine-tuning

北京大学 PEKING UNIVERSITY

- Fine-tuning RoBERTa-Base on the GLUE benchmark:

Algorithm	CoLA	STS-B	MRPC	RTE	SST2	MNLI	QNLI	QQP	Avg
Full Params.	62.07	90.18	92.25	78.34	94.38	87.59	92.46	91.90	86.15
GaLore	61.32	90.24	92.55	77.62	**94.61**	86.92	92.06	90.84	85.77
GoLore@20%	**61.66**	**90.55**	**92.93**	**78.34**	**94.61**	**87.02**	**92.20**	**90.91**	**86.03**

- GoLore shows **superior** performance than GaLore in the above experiments.

Experimental results on pretraining benchmark

北京大学 PEKING UNIVERSITY

LLaMA Pretrain

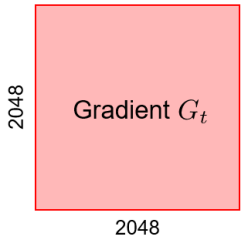
Algorithm	60M	130M	350M	1B
Adam*	34.06 (0.22G)	25.08 (0.50G)	18.80 (1.37G)	15.56 (4.99G)
GaLore*	34.88 (0.14G)	25.36 (0.27G)	18.95 (0.49G)	**15.64** (1.46G)
LoRA*	34.99 (0.16G)	33.92 (0.35G)	25.58 (0.69G)	19.21 (2.27G)
ReLoRA*	37.04 (0.16G)	29.37 (0.35G)	29.08 (0.69G)	18.33 (2.27G)
RSO/GoLore	**34.55**(0.14G)	**25.34** (0.27G)	**18.86** (0.49G)	15.68 (1.46G)
r/d_{model}	128 / 256	256 / 768	256 / 1024	512 / 2048
Training Tokens (B)	1.1	2.2	6.4	13.1

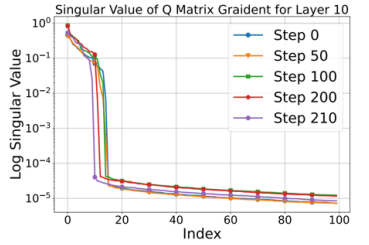
PPL degrades 0.77% Optimizer states saved 70.7%

[1] Yiming Chen, Kun Yuan et al., *A Memory Efficient Randomized Subspace Optimization Method For Training Large Language Models*, ICML 2025

Summary

- LLM memory: parameters, gradients, optimizer states, activations
- LLM gradients exhibit significant low-rank structures
- Core idea: Use low-rank gradients to compute optimizer states, saving memory
- Random Stiefel manifold projection overcomes the bias in SVD projection
- Benefits: <1% performance loss, saves ~70% optimizer memory.

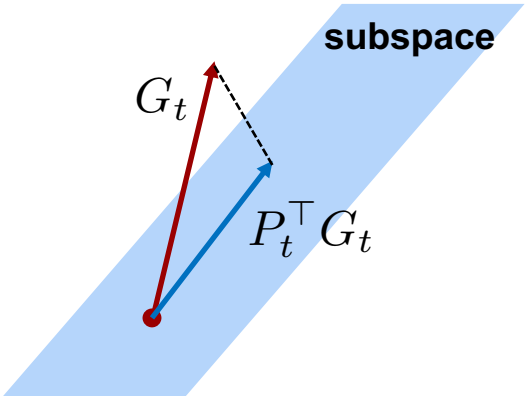




Low-rankness maintained across steps

$$G_t \in \mathbb{R}^{m\times n} \quad P_t^\top G_t \quad g_t = P_t^\top G_t \in \mathbb{R}^{r\times n}$$

Gradient projection



北京大学
PEKING UNIVERSITY

PART 02

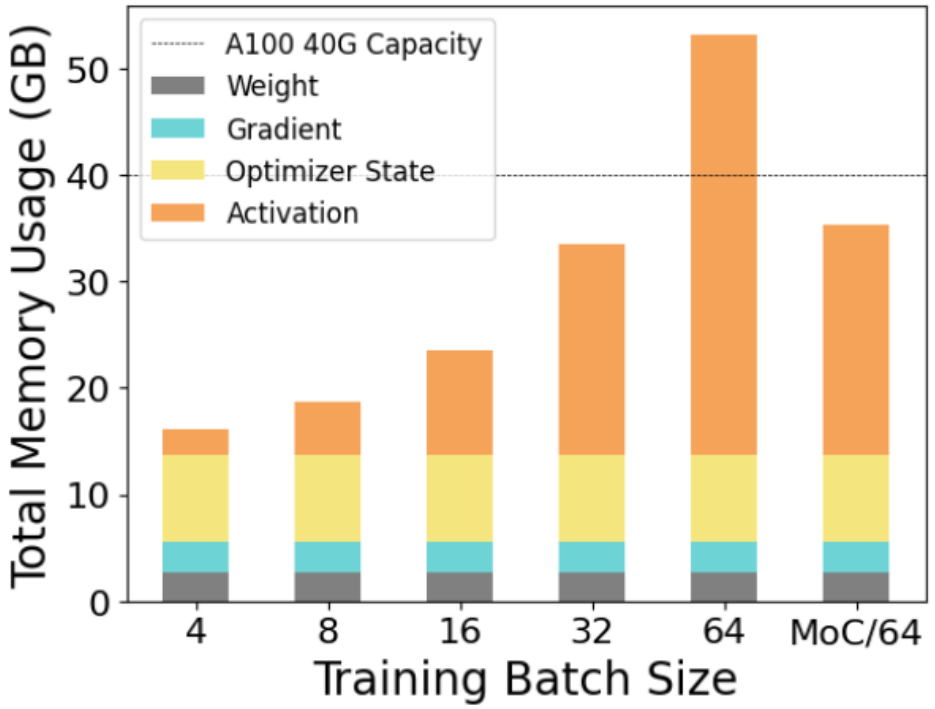
Save Activation via Sparse FFN

T. Wu, Y. He, B. Wang and K. Yuan, *Mixture-of-Channels: Exploiting Sparse FFNs for Efficient LLMs Pre-Training and Inference,* arXiv:2511.09323, 2025

LLM activation memory

LLM Memory= **Param.** + **Grad.** + **Opt. State** + **Activation**

LLaMA-2's memory (context-length 256)



Activations are auxiliary variables stored for gradient computation.

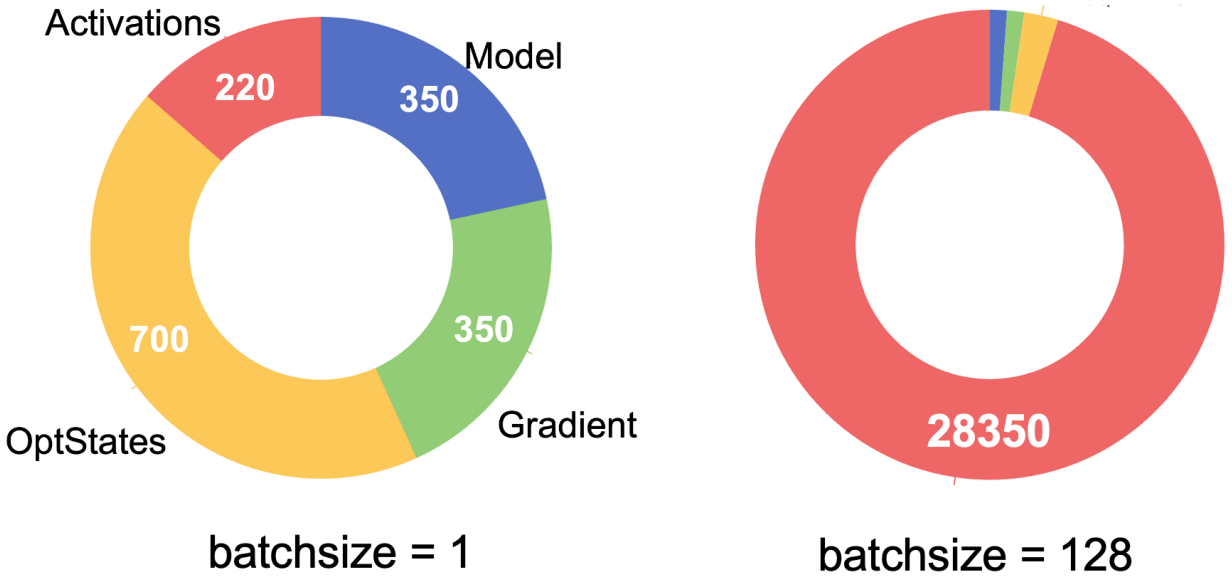
Activation memory is proportional to batch size.

[1] T. Wu, Y. He, B. Wang and K. Yuan, *Mixture-of-Channels: Exploiting Sparse FFNs for Efficient LLMs Pre-Training and Inference,* arXiv:2511.09323, 2025

LLM activation memory

LLM Memory= **Param.** + **Grad.** + **Opt. State**+ **Activation**

GPT3-175B



batchsize = 1

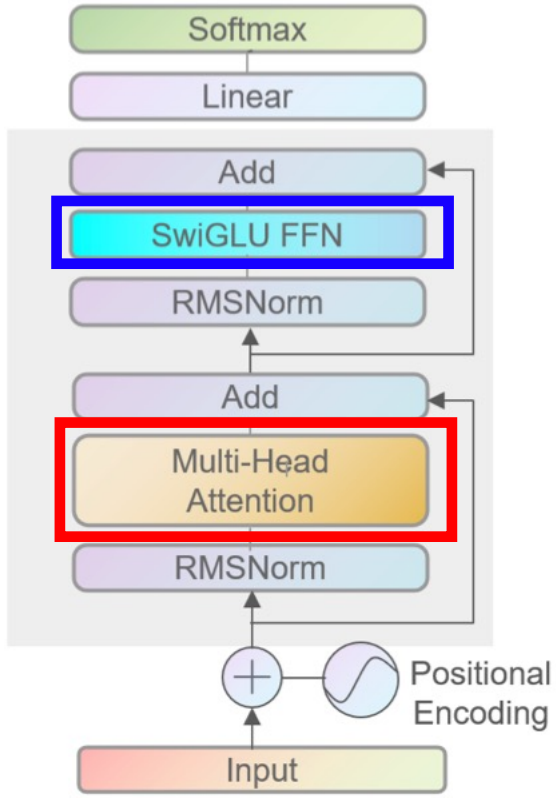
batchsize = 128

When batch size is large, activation memory increases sharply and dominates

Activation memory saving is more critical in large-batch scenarios

This section explores how to save activation memory

Activation memory breakdown



Batch size: b
Context length: s
Hidden dimension: d

Multi-Head Self-Attention (MHSA): for head i, requires:

$$Q_i = XW_Q^i \in \mathbb{R}^{s\times d_h},\ K_i = XW_K^i \in \mathbb{R}^{s\times d_h},\ V_i = XW_V^i \in \mathbb{R}^{s\times d_h}$$

$$A_i = \mathrm{FlashAttention}(Q_i, K_i, V_i) \in \mathbb{R}^{s\times d_h}$$

$$A = [A_1; \cdots; A_h] \in \mathbb{R}^{s\times d},\quad O = AW_o \in \mathbb{R}^{s\times d}$$

Storing $Q, K, V, A, O (5sd)$; $\boldsymbol{5bsd}$ parameters in total for batch size b

SwiGLU FFN: requires computing

$$G = XW_{\text{gate}} \in \mathbb{R}^{s\times d_{\text{ffn}}},\quad U = XW_{\text{up}} \in \mathbb{R}^{s\times d_{\text{ffn}}}$$

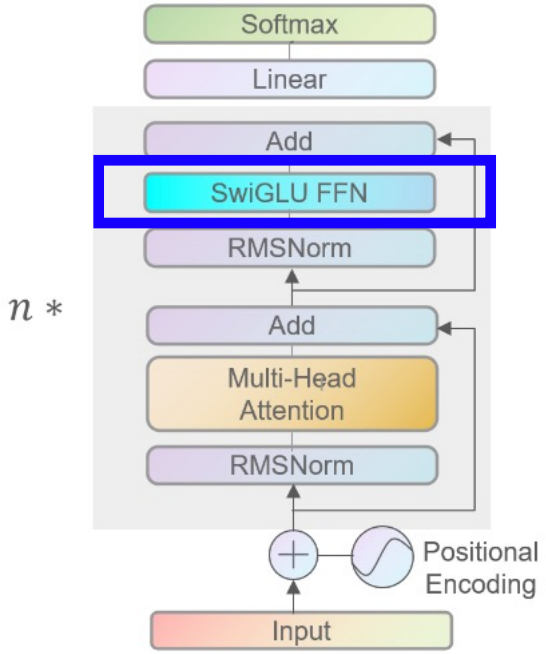
$$S = \mathrm{SiLU}(G) \in \mathbb{R}^{s\times d_{\text{ffn}}},\ Z = S \odot U \in \mathbb{R}^{s\times d_{\text{ffn}}},\ D = ZW_{\text{down}} \in \mathbb{R}^{s\times d}.$$

Storing G, U, S, Z, D with $\boldsymbol{b(d + 4d_{\text{ffn}})s}$ parameters (where $d_{\text{ffn}} = \frac{8}{3}d$)

Activation memory breakdown

- Activation memory: FlashAttention (FA) + FFN + RMSNorm + Residual

$$Layer\ Activation = Attention\ 5bsd + FFN\ 11.67bsd + RMSNorm\ 2bsd + Residual\ 2bsd$$



Activation profiling

		LLaMA (350M)	LLaMA (1.3B)	
Per-Layer (×24)	Attention	177M	336M	
	FFN	400M	791M	(18.54G)
	Others	68M	134M	
LLM head		2.16G	2.16G	
Total		17.64G	32.4G	

11.67/5 = 2.33

FFN activations are **2.314** times of FA's

Saving FFN activations is key to memory reduction

Tong Wu, Kun Yuan, et. al. *Mixture-of-Channels: Exploiting Sparse FFNs for Efficient LLMs Pre-Training and Inference*, arXiv:2511.09323, 2025

SwiGLU

- SwiGLU is widely used in large-model FFNs, e.g., LLaMA-2/3, Qwen2.

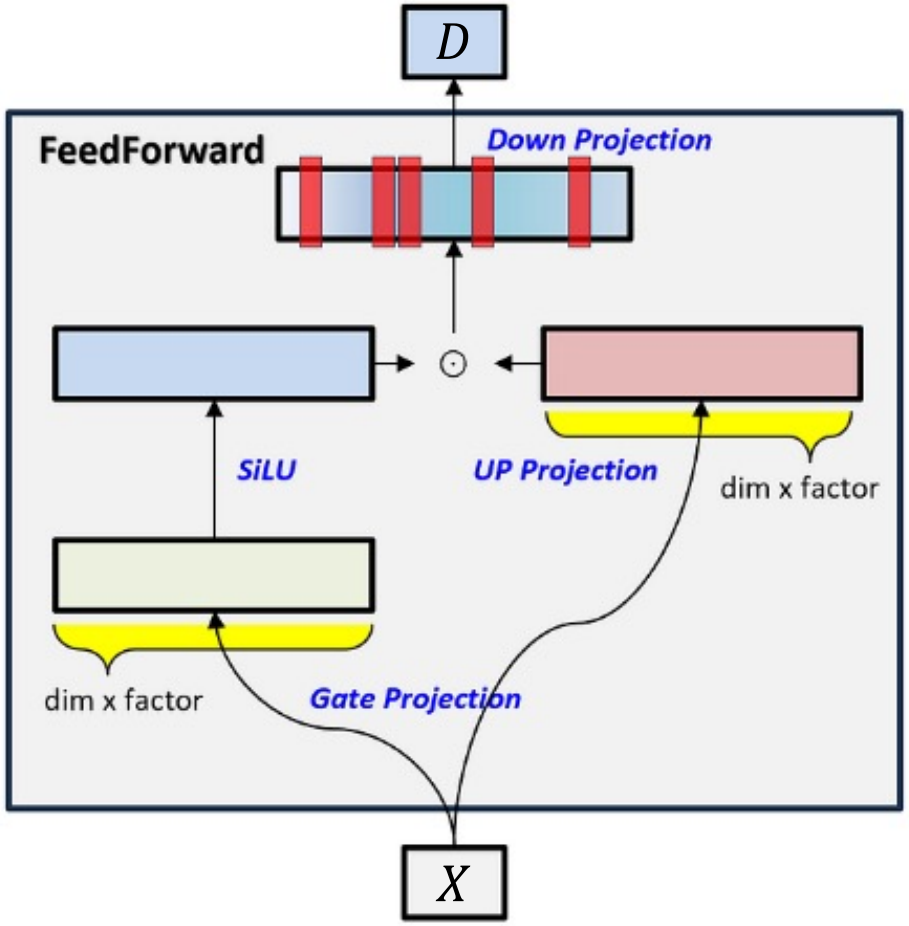
FFN. For each input $X \in \mathbb{R}^{s\times d}$ to the FFN module, we first compute and store:

$$G = XW_{\text{gate}} \in \mathbb{R}^{s\times d_{\text{ffn}}}, \quad U = XW_{\text{up}} \in \mathbb{R}^{s\times d_{\text{ffn}}},$$

where $W_{\text{gate}}, W_{\text{up}} \in \mathbb{R}^{d\times d_{\text{ffn}}}$ are the weights corresponding to the gating and up-sampling branches in the SwiGLU activation.

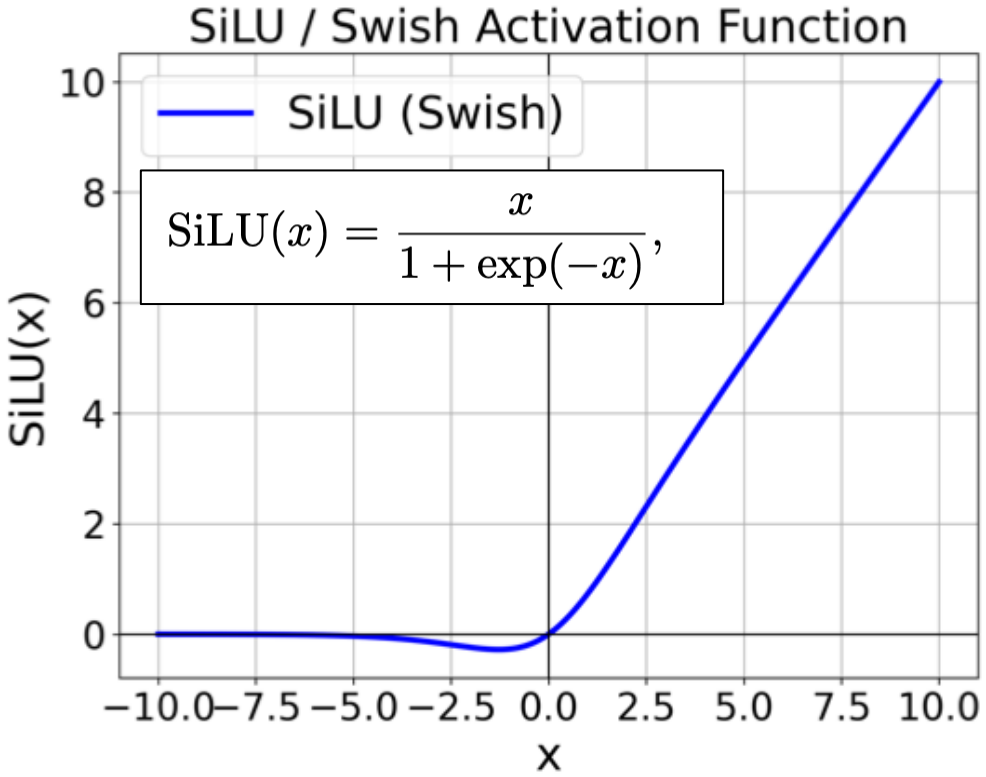
$$S = \text{SiLU}(G) \in \mathbb{R}^{s\times d_{\text{ffn}}}$$
$$Z = S \odot U \in \mathbb{R}^{s\times d_{\text{ffn}}}$$
$$D = ZW_{\text{down}} \in \mathbb{R}^{s\times d},$$

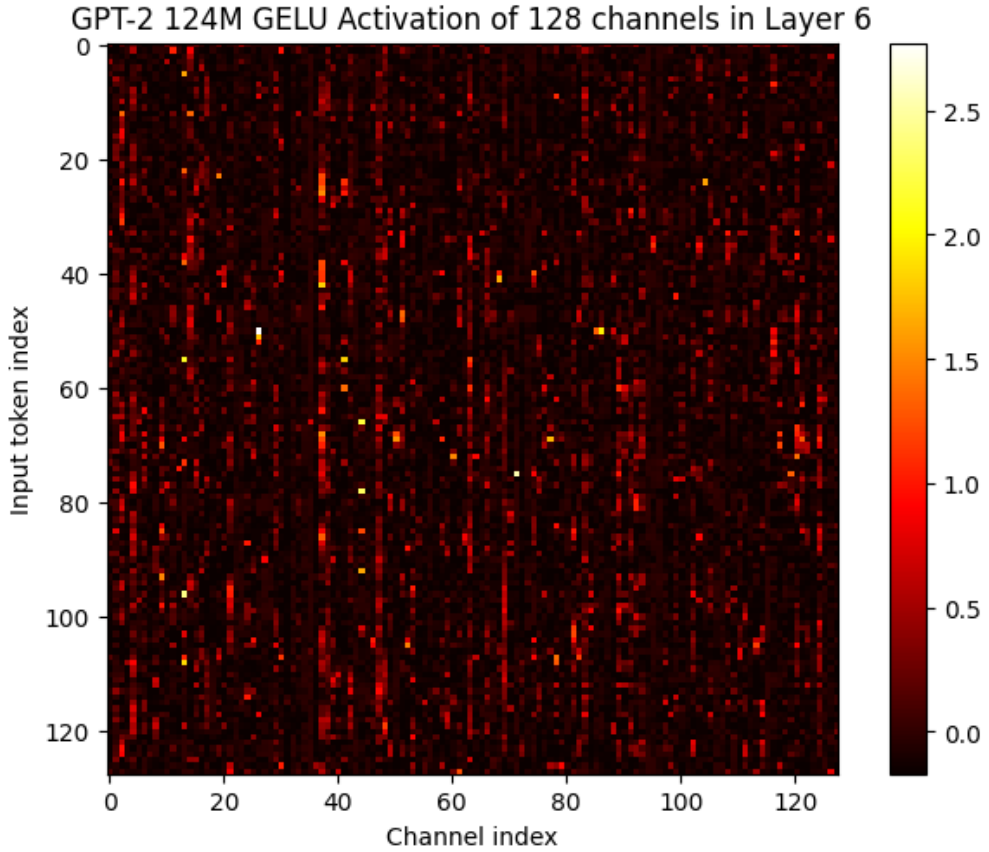
where $W_{\text{down}} \in \mathbb{R}^{d_{\text{ffn}}\times d}$ is the down-sampling weight.



The structure of LLM's FFN: SwiGLU

Tong Wu, Kun Yuan, et. al. *Mixture-of-Channels: Exploiting Sparse FFNs for Efficient LLMs Pre-Training and Inference*, arXiv:2511.09323, 2025





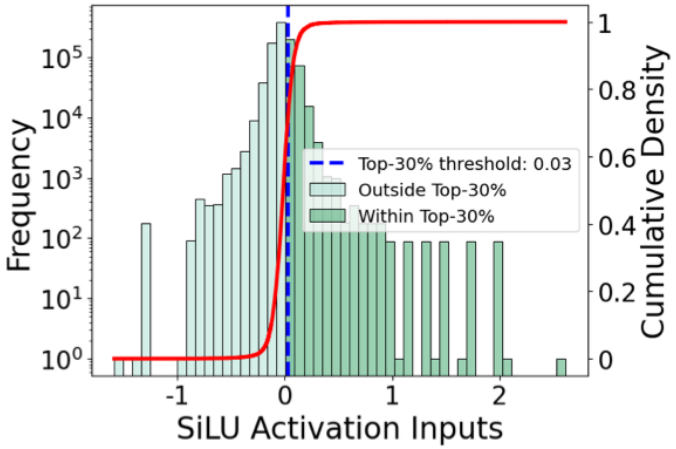
- When $x \geq 0$, SiLU produces strong activations;
- When $x < 0$, SiLU suppresses the input signal;

>70% are negative or near 0 in each row

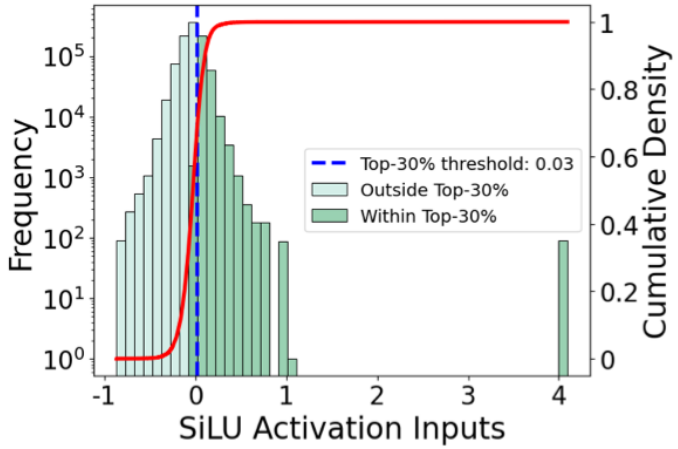
The structure of LLM's FFN: SwiGLU

北京大学 PEKING UNIVERSITY

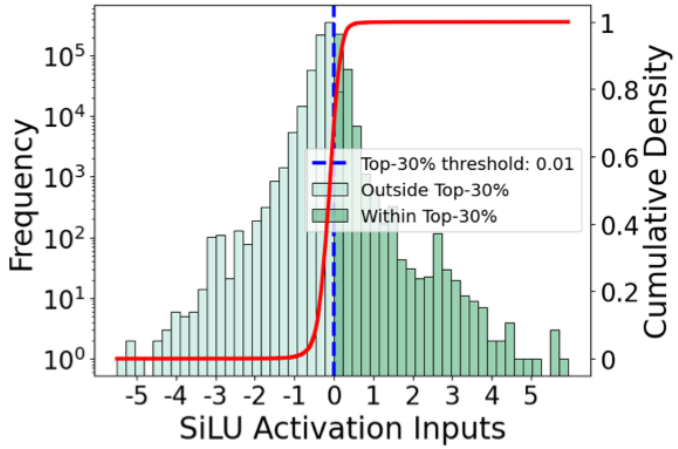
~70% inputs of SiLU are below 0



(a) LLaMA-2 Layer 0.

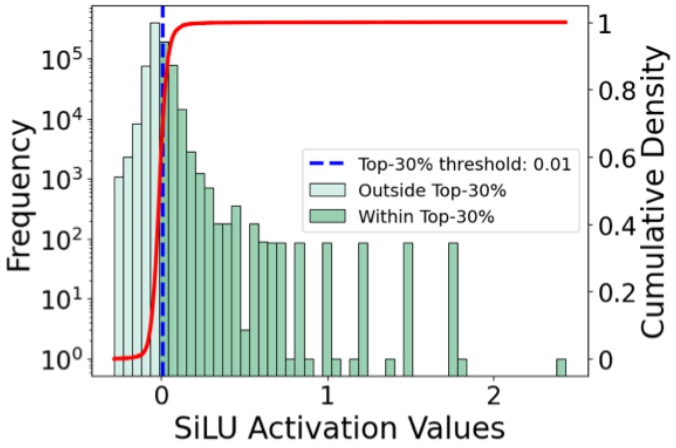


(b) LLaMA-2 Layer 16.

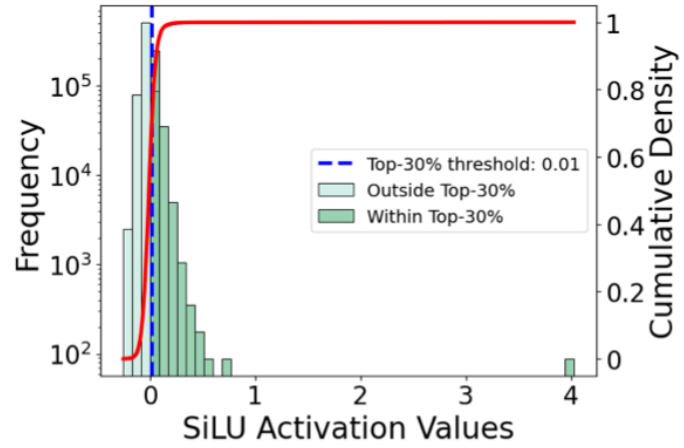


(c) LLaMA-2 Layer 31.

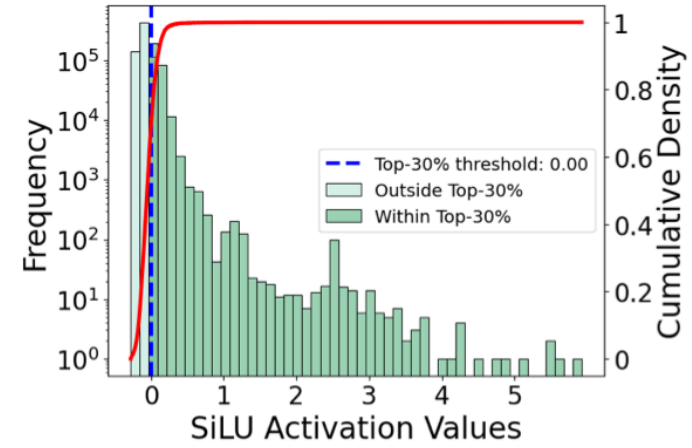
~70% outputs of SiLU are suppressed, only the remaining 30% are activated



(d) LLaMA-2 Layer 0.



(e) LLaMA-2 Layer 16.



(f) LLaMA-2 Layer 31.

New FFN structure based on sparsity: Mixture of Channels

北京大学 PEKING UNIVERSITY

Core idea：for every input token, adaptively select Top-K channels

Traditional FFN structure: SwiGLU

FFN. For each input $X \in \mathbb{R}^{s\times d}$ to the FFN module, we first compute and store:

$$G = XW_{\text{gate}} \in \mathbb{R}^{s\times d_{\text{ffn}}}, \quad U = XW_{\text{up}} \in \mathbb{R}^{s\times d_{\text{ffn}}},$$

where $W_{\text{gate}}, W_{\text{up}} \in \mathbb{R}^{d\times d_{\text{ffn}}}$ are the weights corresponding to the gating and up-sampling branches in the SwiGLU activation.

$$S = \text{SiLU}(G) \in \mathbb{R}^{s\times d_{\text{ffn}}}$$
$$Z = S \odot U \in \mathbb{R}^{s\times d_{\text{ffn}}}$$
$$D = ZW_{\text{down}} \in \mathbb{R}^{s\times d},$$

where $W_{\text{down}} \in \mathbb{R}^{d_{\text{ffn}}\times d}$ is the down-sampling weight.

New FFN structure based on sparsity: Mixture of Channels

MoC. For each input $X \in \mathbb{R}^{s\times d}$ to the FFN module, we first compute and store:

$$G = XW_{\text{gate}} \in \mathbb{R}^{s\times d_{\text{ffn}}}, \quad U = XW_{\text{up}} \in \mathbb{R}^{s\times d_{\text{ffn}}},$$

where $W_{\text{gate}}, W_{\text{up}} \in \mathbb{R}^{d\times d_{\text{ffn}}}$ are the weights corresponding to the gating and up-sampling branches in the SwiGLU activation.

$$S = \text{SiLU}(G), \quad S' = S \odot M,$$
$$Z' = S' \odot U,$$
$$D = Z'W_{\text{down}},$$

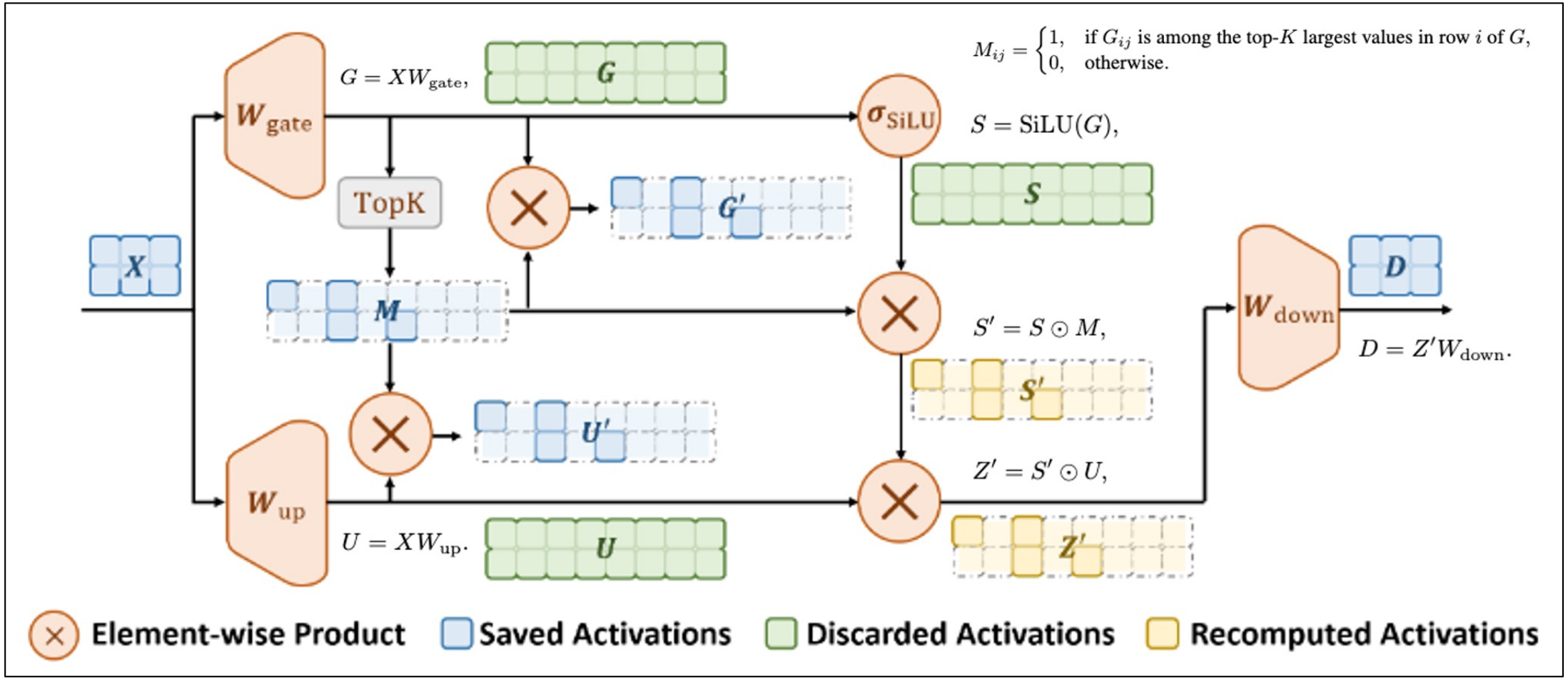
where W_{down} is the down-sampling weight, and M is defined as

$$M_{ij} = \begin{cases} 1, & \text{if } G_{ij} \text{ is among the top-}K \text{ largest values in row } i \text{ of } G, \\ 0, & \text{otherwise.} \end{cases}$$

Since only part of the channels are activated, we call it a Mixture of Channels (MoC) model.

Mixture of Channels (MoC) model

- Schematic of the MoC architecture (typically selecting the top 20% of channels).



Efficient training of Mixture of Channels models

北京大学 PEKING UNIVERSITY

MoC's forward pass

$$G = XW_{\text{gate}}, \qquad U = XW_{\text{up}}.$$
$$M = \text{TopK}(G)$$
$$S = \text{SiLU}(G), \quad S' = S \odot M,$$
$$Z' = S' \odot U, \quad D = Z'W_{\text{down}}.$$

MoC's backward pass

$$\nabla W_{\text{down}} = (Z')^\top \nabla D, \qquad \nabla Z' = \nabla D W_{\text{down}}^\top,$$
$$\nabla S' = (U \odot M) \odot \nabla Z', \qquad \nabla U = S' \odot \nabla Z',$$
$$\nabla S = \nabla S', \qquad \nabla G = \nabla S' \odot (\nabla \text{SiLU})(G),$$
$$\nabla W_{\text{gate}} = X^\top \nabla G, \qquad \nabla W_{\text{up}} = X^\top \nabla U,$$
$$\nabla X = \nabla G W_{\text{gate}}^\top + \nabla U W_{\text{up}}^\top,$$

Sparse storage

- Store sparse activations $Z' = Z \odot M$, $U' = U \odot M$ and $S' = S \odot M$
- Since $\nabla S' = \nabla S = U \odot M \odot \nabla Z'$, ∇SiLU operates coordinate-wise, only $G' = G \odot M$ is stored
- In contrast, traditional FFN architectures require storing dense matrices Z, U, S, G

Efficient training of Mixture of Channels models

北京大学 PEKING UNIVERSITY

MoC's activation computation

	MoC	MoC+GCP	FFN	FFN+GCP
Stored Activations	$G \odot M$ $U \odot M$ $S \odot M$ $Z \odot M$ M and D	$G \odot M$ $U \odot M$ – – M and D	G U S Z D	G U – – D
Memory Cost	$bsd_{\text{ffn}} + bsd$ $(3.67bsd)$	$0.6bsd_{\text{ffn}} + bsd$ $(2.6bsd)$	$4bsd_{\text{ffn}} + bsd$ $(11.67bsd)$	$2bsd_{\text{ffn}} + bsd$ $(6.33bsd)$

- MoC only stores **sparse activations**
- MoC saves ~**68%** activation memory
- S and Z can be easily recomputed instead of being stored

MoC system-level operator optimization and memory savings

- Unstructured sparsity does not reduce computation, and top-K introduces extra overhead
- With optimization using RAFT and Triton, MoC achieves FFN-equivalent computation time
- 2:8 structured sparse operators can achieve computational savings.

MoC does not introduce significant overall computational overhead

	Standard FFN (ms)	MoC using optimized kernels (ms)
Forward	20.2	21.1
Backward	21.6	22.8
Total	41.8	43.9

Pretrain LLaMA on C4

	60M	130M	350M	1B
FFN-based LLM + AdamW	30.44 (38.3G)	23.92 (54.1G)	18.26 (52.5G)	15.56 (56.6G)
GaLore	34.88 (38.1G)	25.36 (53.9G)	18.95 (51.7G)	15.64 (52.5G)
SLTrain	34.15 (33.6G)	26.04 (59.4G)	19.42 (69.1G)	16.14 (70.0G)
MoC	**30.59** (21.8G)	**24.02** (41.7G)	**18.57** (34.6G)	15.80 (41.9G)
$\text{MoC}_{2:8}$	31.02 (22.1G)	24.12 (42.3G)	18.68(36.4G)	**15.62** (42.7G)
batch size per GPU	256	256	128	64
K/d_{model}	128 / 256	384 / 768	512 / 1024	1024 / 2048
Training Tokens	1.1B	2.2B	6.4B	13.1B

With only a **0.4%** loss increase, end-to-end total memory is reduced by **24.6%** (all memory accounted for).

Extending MoC to other LLM architectures

北京大学 PEKING UNIVERSITY

Compatibility of MoC with **MoE** architecture

	160M	530M
Vanilla Mixtral	**23.77** (48.3G)	**18.65** (41.7G)
Mixtral+MoC	**24.44** (38.2G)	**18.88** (30.0G)
batch size per GPU	256	128
K/d_{model}	256 / 512	512 / 1024

Compatibility of MoC with **GQA** architecture

Model	Structure	Ppl.	Memory
LLaMA-130M	GQA	24.02	53.9G
LLaMA-130M	GQA+MoC	24.26	**34.4G**
LLaMA-350M	GQA	18.51	52.9G
LLaMA-350M	GQA+MoC	18.69	**36.9G**

Compatibility of MoC with **Qwen** architecture

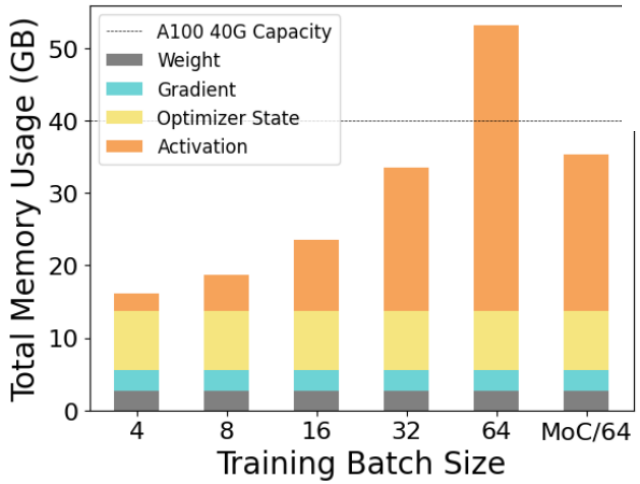
Model	Ppl.
Qwen3-300M	18.52
Qwen3-300M+MoC	18.59

Compatibility of MoC with **LLaMA-7B**

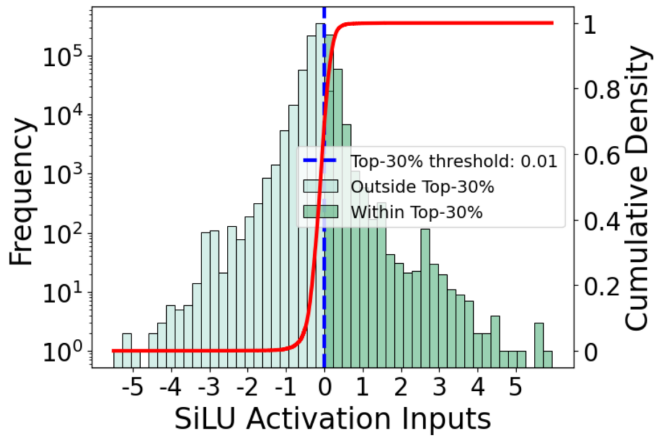
Model	Ppl.
LLaMA-7B	26.14
LLaMA-7B+MoC	26.47

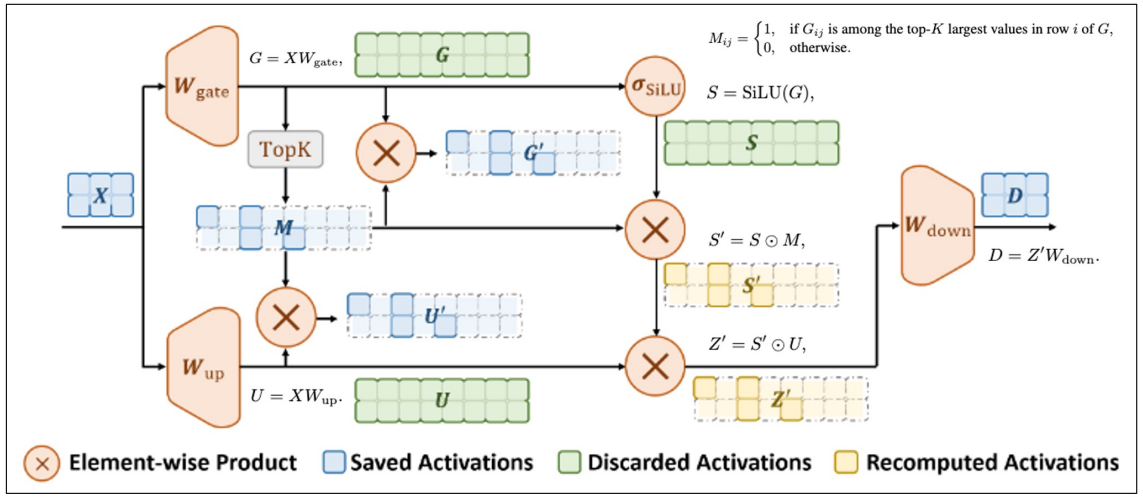
Summary

- In LLM memory, **activations** dominate; within activations, **FFN memory** usage is dominant
- FFN: SiLU activation is sparse; most channels suppressed



- Leveraging SiLU sparsity, we design MoC to save memory and speed up inference
- Benefits: <1% performance loss, saves ~68% activation memory, ~25% total memory, >10% faster inference.





北京大学
PEKING UNIVERSITY

PART 03

Save Model and Gradient via Cross-Layer Structure

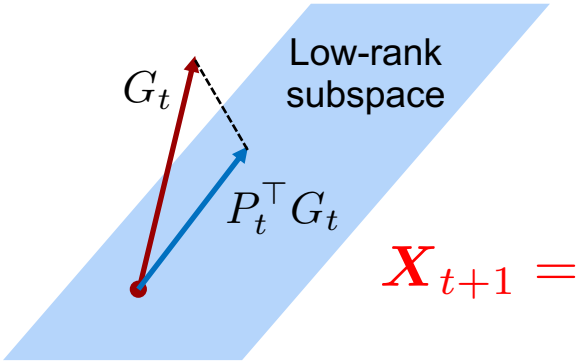
Boao Kong, Junzhu Liang, Yuxi Liu, Renjia Deng, Kun Yuan, “CR-Net: Scaling Parameter-Efficient Training with Cross-Layer Low-Rank Structure”, arXiv:2509.18993, 2025

Memory overhead in LLMs pre-training

北京大学 PEKING UNIVERSITY

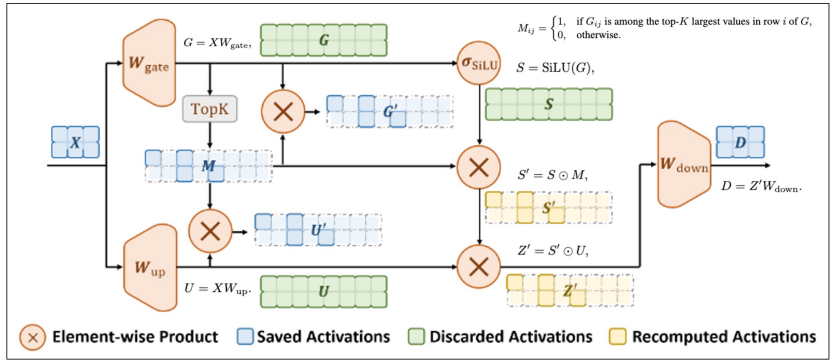
Memory = **Parameter** + **Gradient** + **Optimizer states** + **Activation**

- Low-rank projection only saves the memory of **Optimizer states**



$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \boldsymbol{P}_t \boldsymbol{\rho}(\boldsymbol{P}_t^\top \boldsymbol{G}_t)$$

- Sparse activation in MoC onlys saves the memory of **Activation**



$$M = \text{TopK}(G)$$
$$S = \text{SiLU}(G),$$
$$S' = S \odot M,$$
$$Z' = S' \odot U,$$

- How to save the **Parameter** and **Gradient**? **Parameter-efficient Methods!**
- Parameter-efficient methods can save the memory of parameter, gradient, and optimizer states at the same time.

Bottleneck of LoRA in LLMs pre-training

北京大学 PEKING UNIVERSITY

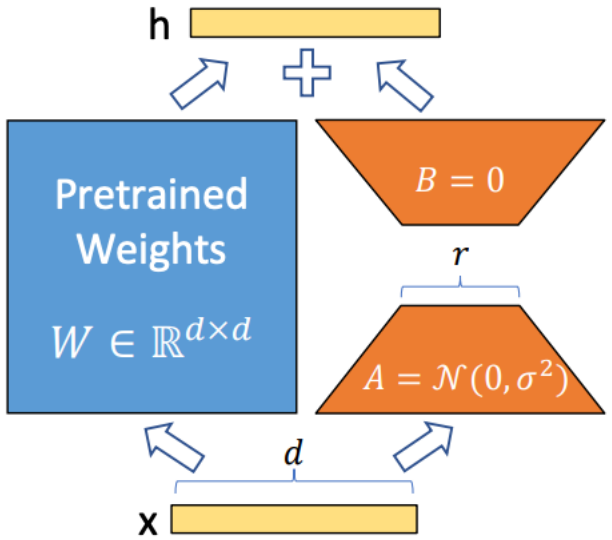
- Although LoRA enables fine-tuning of LLMs with fewer parameters, it is not applicable in pre-training.

$$\min_{W \in \mathbb{R}^{p \times q}} \quad \mathbb{E}_{\xi \sim \mathcal{D}}[F(W; \xi)] \qquad \text{(loss function for pre-training)}$$

$$\min_{A \in \mathbb{R}^{p \times r}, B \in \mathbb{R}^{r \times q}} \quad \mathbb{E}_{\xi \sim \mathcal{D}}[F(W + AB; \xi)] \qquad \text{(loss function for LoRA)}$$

Update A, B

Fix X



Pre-trian LLaMA on C4

Methods	60M		130M		350M		1B	
	Perplexity	Memory	Perplexity	Memory	Perplexity	Memory	Perplexity	Memory
AdamW♠	34.06	0.36G	25.08	0.76G	18.80	2.06G	15.56	7.80G
Low-Rank♠	78.18	0.26G	45.51	0.54G	37.41	1.08G	142.53	3.57G
LoRA♠	34.99	0.36G	33.92	0.80G	25.58	1.76G	19.21	6.17G

- The pre-training PPL of LoRA increases significantly.
- Calls for an effective algorithm for pre-training.

Low-Rank Activation Residuals Between Adjacent Layers in LLMs

- Parameters lack low-rank structure; low-rank approximation fails

$$Q_\ell = XW_\ell^Q \approx XA_\ell^Q B_\ell^Q$$

Activation Input Parameter Low-rank parameter

- **Core idea**: Compensate approximation error using **previous layer's activations**[1]

$$Q_\ell = XW_\ell^Q \approx Q_{\ell-1} + XA_\ell^Q B_\ell^Q$$

Activation in layer ℓ

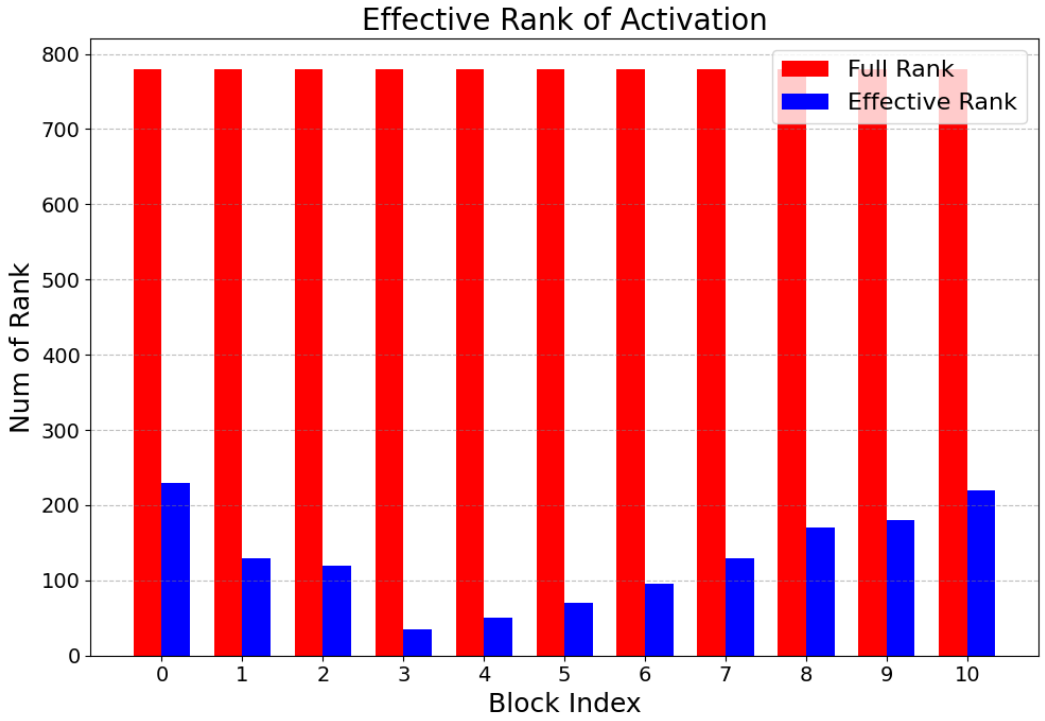
Activation in layer $\ell - 1$

In other words, adjacent-layer activation residuals $Q_\ell - Q_{\ell-1}$ exhibit low-rank structure

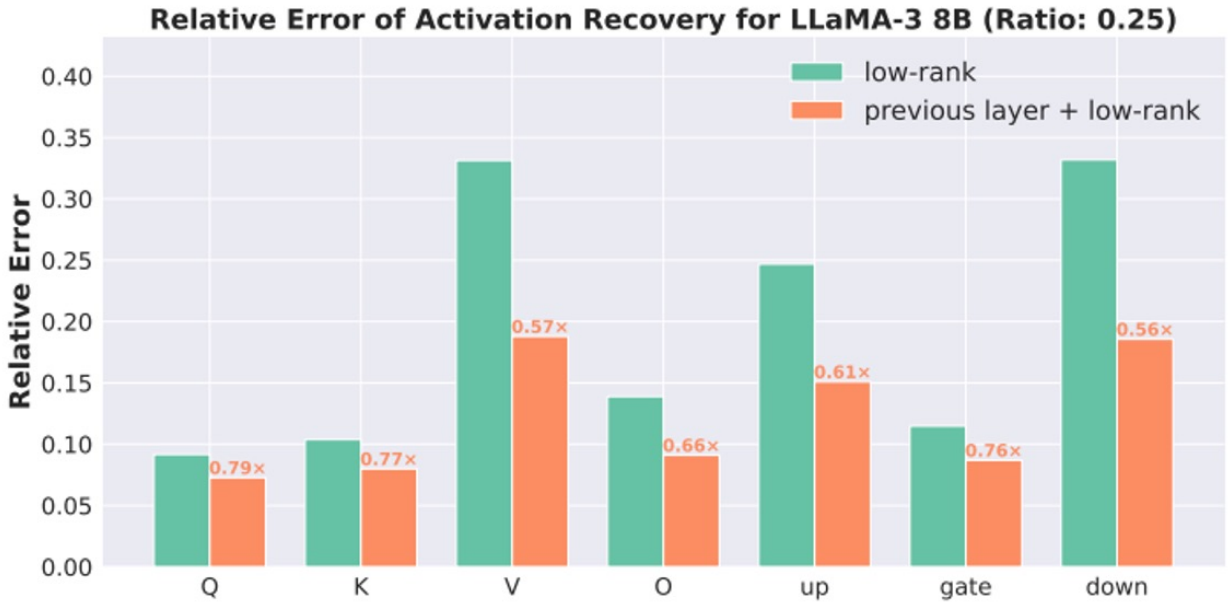
[1] Boao Kong, Kun Yuan et al. *CR-Net: Scaling Parameter-Efficient Training with Cross-Layer Low-Rank Structure*, arXiv:2509.18993, 2025

Low-Rank Activation Residuals Between Adjacent Layers in LLMs

$$Q_\ell = XW_\ell^Q \approx XA_\ell^Q B_\ell^Q$$
$$Q_\ell = XW_\ell^Q \approx Q_{\ell-1} + XA_\ell^Q B_\ell^Q$$



Activation residuals between layers exhibit low-rank property



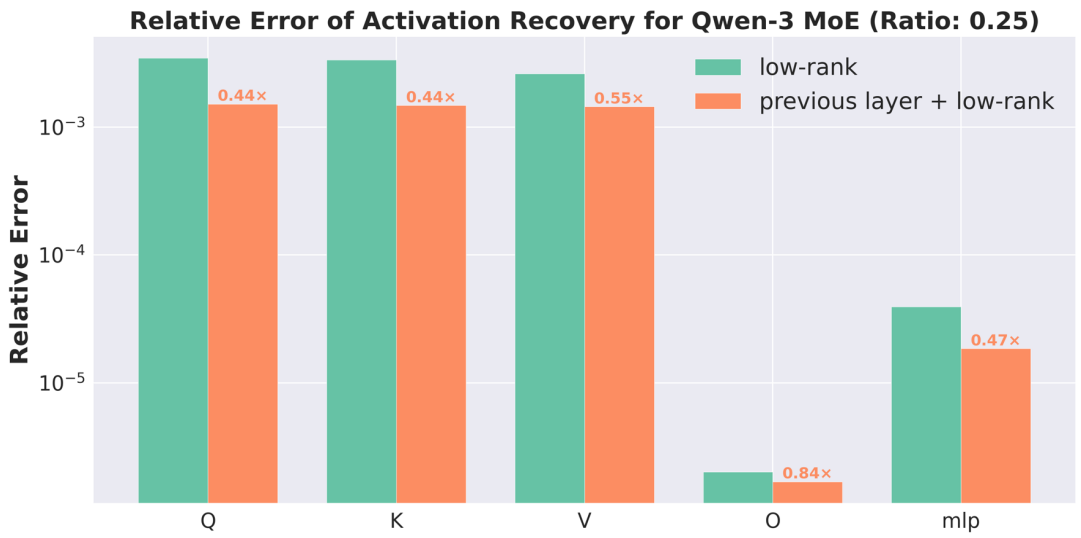
After error compensation, recovery error is reduced by **9%~54%**

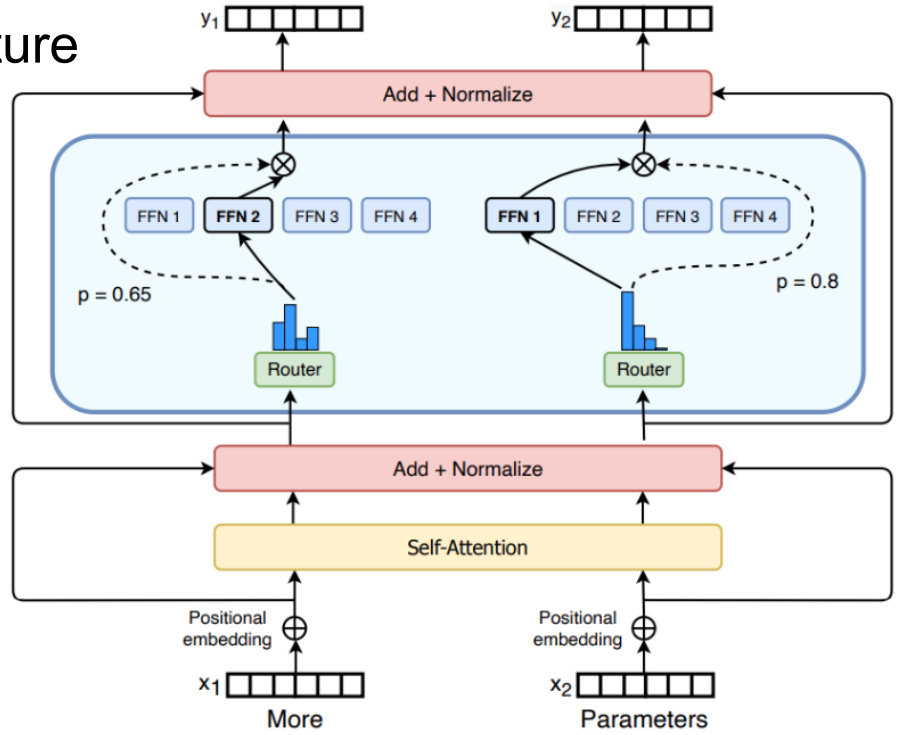
[1] Boao Kong, Kun Yuan et al. *CR-Net: Scaling Parameter-Efficient Training with Cross-Layer Low-Rank Structure*, arXiv:2509.18993, 2025

Low-Rank Activation Residuals Between Adjacent Layers in LLMs

北京大学 PEKING UNIVERSITY

- Similar cross-layer low-rank structure in **MoE** architecture





- With the same rank, compensating with the previous layer's activations reduces the error by **16%~56%**

[1] Boao Kong, Kun Yuan et al. *CR-Net: Scaling Parameter-Efficient Training with Cross-Layer Low-Rank Structure*, arXiv:2509.18993, 2025

Theories behind low-rank structure in cross-layer activations

北京大学 PEKING UNIVERSITY

- The (high cosine) similarity between adjacent layer activations makes their residuals low-rank

Assumption 1 (Cosine similarity of adjacent attentions). *For $l = 2, 3, \cdots, L$, let $Y_l^P \in \mathbb{R}^{d\times d}$ as the activation of the linear of position P for the l-th layer. There exists a constant $\epsilon \in (0, 1)$ such that:*

$$\frac{\langle Y_l^P, Y_{l-1}^P \rangle_F}{\|Y_l^P\|_F \cdot \|Y_{l-1}^P\|_F} \geq 1 - \epsilon,$$

where $\langle \cdot, \cdot \rangle_F$ denotes the inner production of matrices induced by Frobenius norm.

Theorem 1. *Suppose Assumption 1 holds. Then there exists $r_0 > 0$ such that the approximation $\widetilde{Y}_{l,\beta}^P$ obtained by Eq. (3) has a lower error than the direct low-rank approximation $LR_r(Y_l^P)$ by a properly-selected β if $r < r_0$. Specifically, it holds that:*

$$\left\| Y_l^P - \widetilde{Y}_{l,\beta}^P \right\|_F^2 \leq \left\| Y_l^P - LR_r(Y_l^P) \right\|_F^2 .$$

Low-Rank Approximation with Error Compensation

Direct Low-Rank Approximation

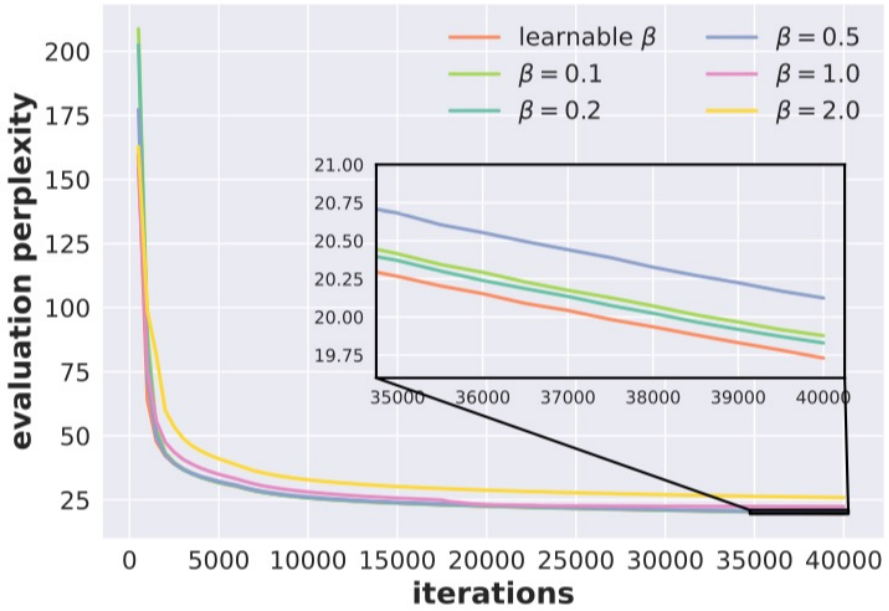
Cross-layer low-rank structure introduces smaller error than direct low-rank approximation of activations.

Learnable scaling factors

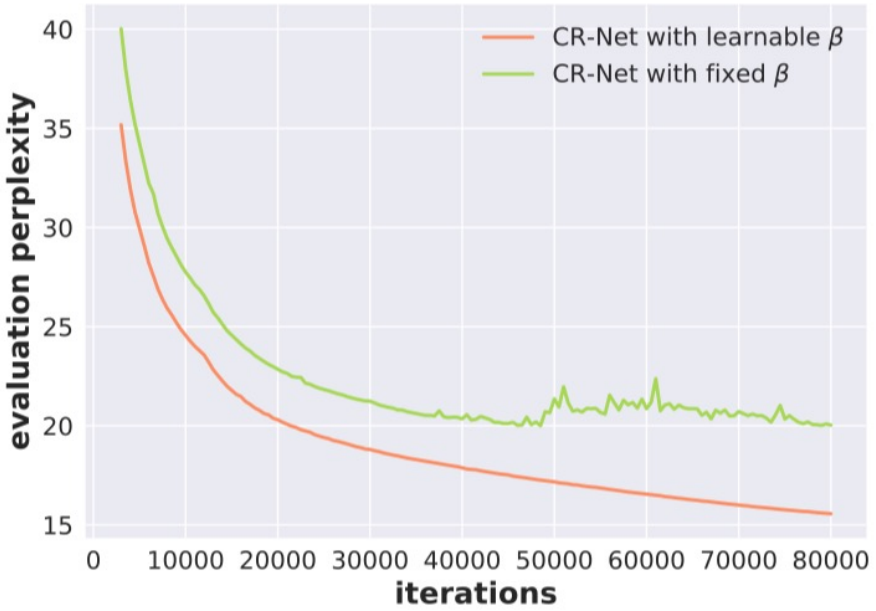
$$Y_\ell^{\mathrm{P}} = \beta_\ell^{\mathrm{P}} Y_{\ell-1}^{\mathrm{P}} + X_\ell^{\mathrm{P}} A_\ell^{\mathrm{P}} B_\ell^{\mathrm{P}}$$

- β_0 and β_l balance information between previous and current layers
- We make β_l **learnable** to **dynamically** adjust the influence of historical activations and low-rank output

LLaMA-2 350M



LLaMA-2 1B



Cross-layer Low-Rank Residual Network (CR-Net)

北京大学 PEKING UNIVERSITY

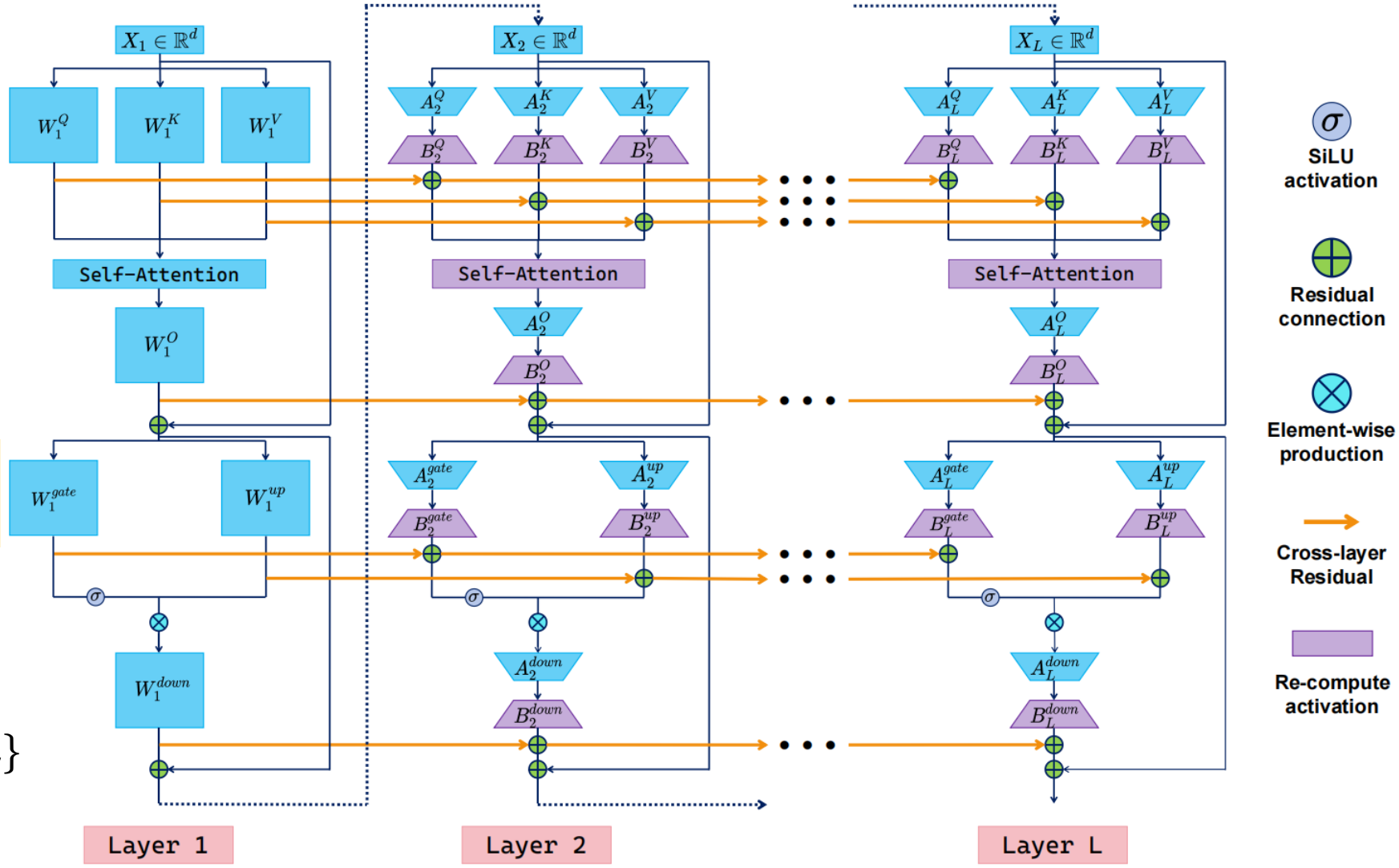
- Full-rank parameters in the firsr transformer layer
- For layer $\ell = 2, 3, \cdots, L$

$$Y_\ell^{\mathrm{P}} = \beta_\ell^{\mathrm{P}} Y_{\ell-1}^{\mathrm{P}} + X_\ell^{\mathrm{P}} A_\ell^{\mathrm{P}} B_\ell^{\mathrm{P}}$$

Learnable factor

Activation at position P

$$\mathrm{P} \in \{Q, K, V, O, gate, up, down\}$$



Boao Kong, Kun Yuan et al. *CR-Net: Scaling Parameter-Efficient Training with Cross-Layer Low-Rank Structure*, submitted to NeurIPS 2025

Cross-layer Low-Rank Residual Network (CR-Net)

北京大学 PEKING UNIVERSITY

- Full-size pre-training

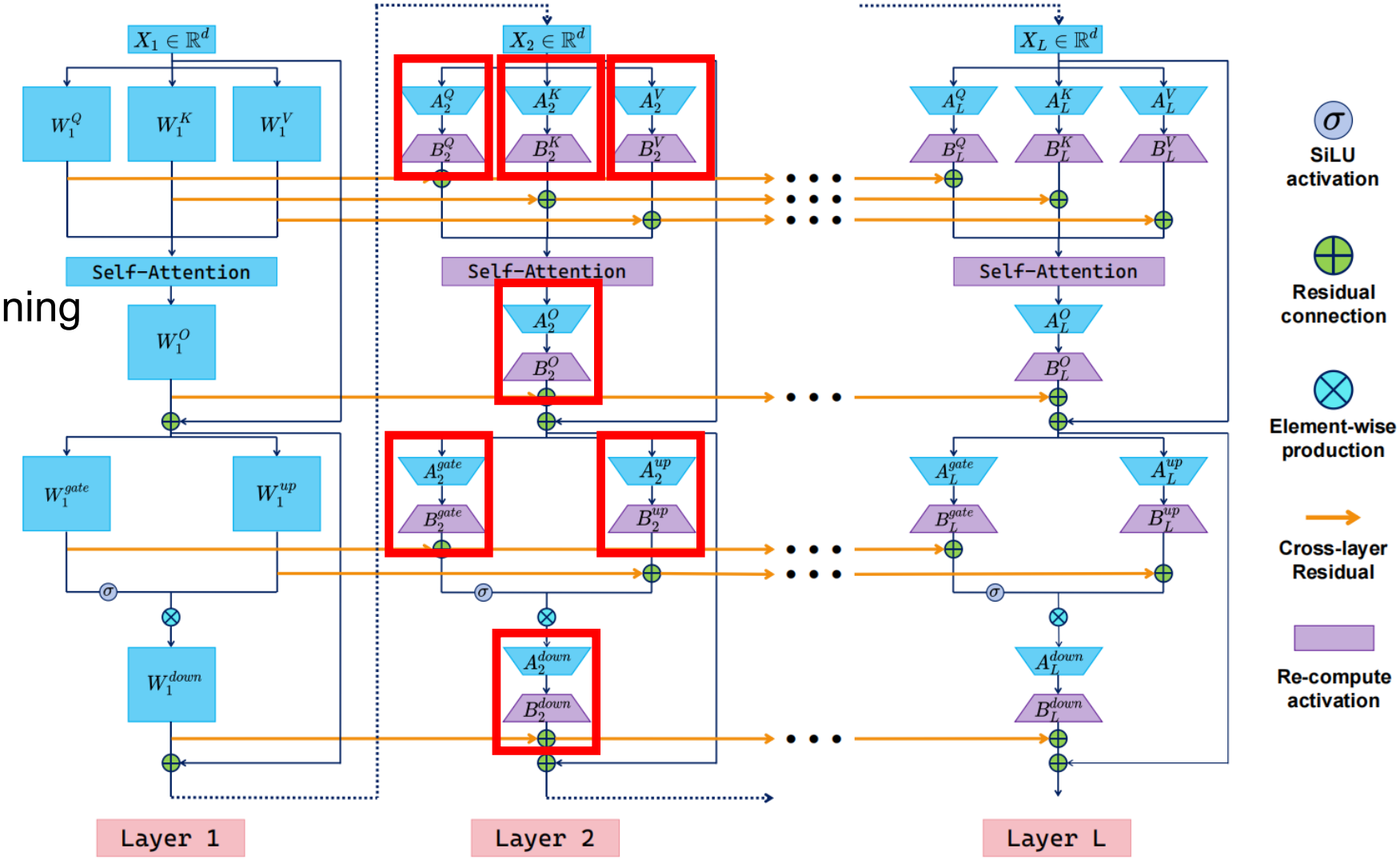
$$Y_\ell^{\mathrm{P}} = X_\ell^{\mathrm{P}} W_\ell^{\mathrm{P}} \quad \textbf{(m, n)}$$

- Parameter-efficient pre-training

$$Y_\ell^{\mathrm{P}} = \beta_\ell^{\mathrm{P}} Y_{\ell-1}^{\mathrm{P}} + X_\ell^{\mathrm{P}} A_\ell^{\mathrm{P}} B_\ell^{\mathrm{P}}$$

(m, r), (r, n)

- Reduce parameter from mn to $r(m+n)$
- **Leading to a smaller model, gradient, and optimizer states**



Boao Kong, Kun Yuan et al. *CR-Net: Scaling Parameter-Efficient Training with Cross-Layer Low-Rank Structure*, submitted to NeurIPS 2025

CR-Net: Experimental results

Pretrain LLaMA on C4

Approach	60M			130M			350M			1B		
	1.1B tokens			2.2B tokens			6.4B tokens			13.1B tokens		
	PPL	Para	Mem	PPL	Para	Mem	PPL	Para	Mem	PPL	Para	Mem
Full-rank	34.06	58	0.43	24.36	134	1.00	**18.80**	368	2.74	15.56	1339	9.98
LoRA	34.99	58	0.37	33.92	134	0.86	25.58	368	1.94	19.21	1339	6.79
ReLoRA	37.04	58	0.37	29.37	134	0.86	29.08	368	1.94	18.33	1339	6.79
SLTrain	34.15	44	0.32	26.04	97	0.72	19.42	194	1.45	16.14	646	4.81
CoLA	34.04	43	0.32	24.48	94	0.70	19.40	185	1.38	15.52	609	4.54
CR-Net ◊	**32.76**	43	0.32	**24.31**	90	0.67	18.95	183	1.36	**15.28**	583	4.35
GaLore	34.88	58	0.36	25.36	134	0.79	18.95	368	1.90	15.64	1339	6.60
RSO	34.55	58	0.36	25.34	134	0.79	18.87	368	1.90	15.86	1339	6.60
Apollo	**31.55**	58	0.36	**22.94**	134	0.79	**16.85**	368	1.90	14.20	1339	6.60
CR-Net †	32.76	43	0.32	23.74	106	0.79	17.08	250	1.86	**14.05**	870	6.48

- CR-Net achieves lower loss with fewer parameters (~43.6% in LLaMA-2 1B)
- At equal memory, CR-Net achieves lower loss at larger scales

The recomputation strategy of CR-Net

北京大学 PEKING UNIVERSITY

- CR-Net cannot directly save activation

(s, m) x (m, n) = (s, n) : Lsn

Vanilla: $Y_\ell^{\mathrm{P}} = X_\ell^{\mathrm{P}} W_\ell^{\mathrm{P}}$

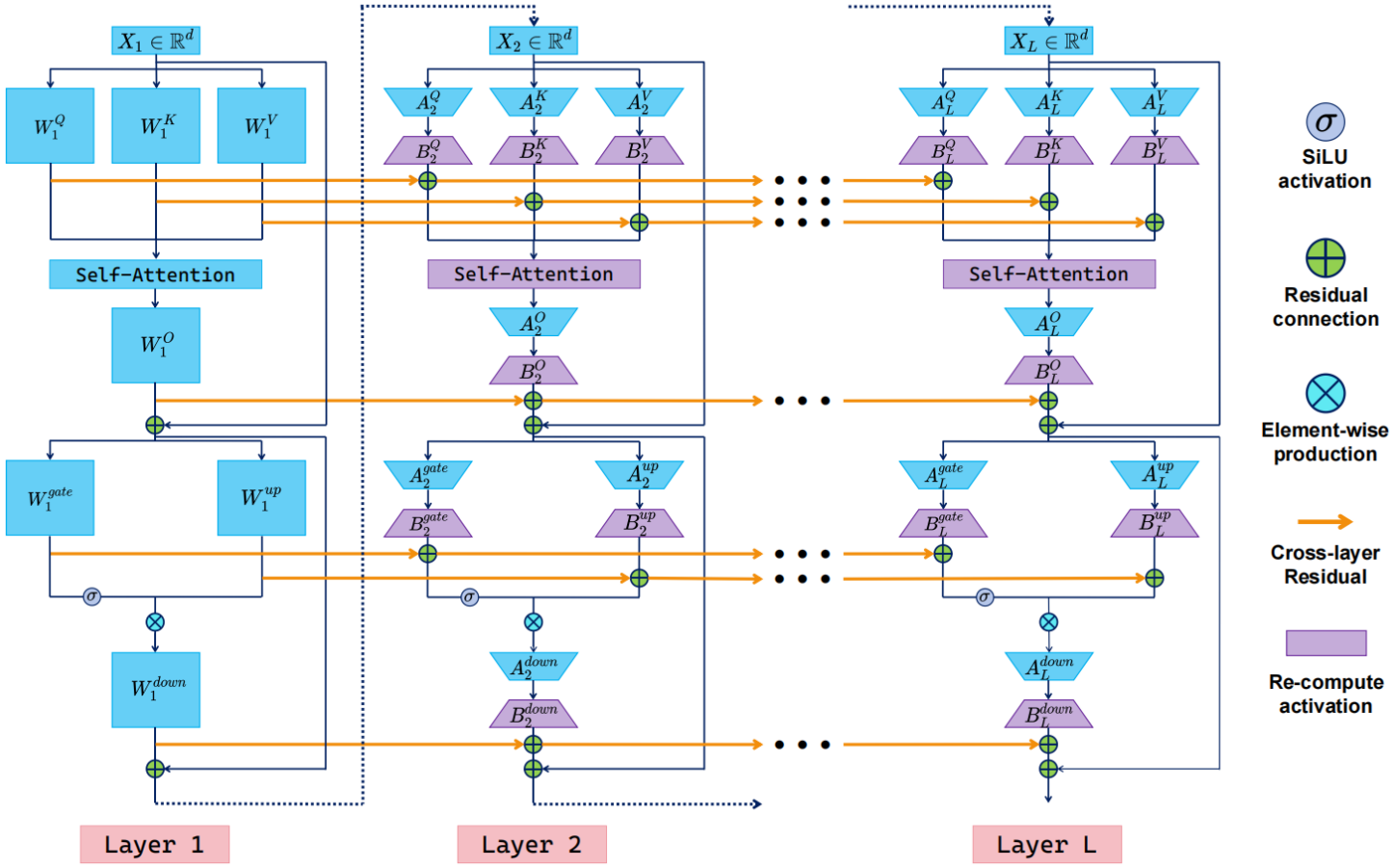
CR-Net: $Y_\ell^{\mathrm{P}} = \beta_\ell^{\mathrm{P}} Y_{\ell-1}^{\mathrm{P}} + X_\ell^{\mathrm{P}} A_\ell^{\mathrm{P}} B_\ell^{\mathrm{P}}$

(s, m) x (m, r) x (r, n) = (s, n): Lsn

- CR-Net + recomputation:

$$Y_\ell^{\mathrm{P}} = \frac{1}{\beta_\ell^{\mathrm{P}}}\left(Y_{\ell+1}^{\mathrm{P}} - X_{\ell+1}^{\mathrm{P}} A_{\ell+1}^{\mathrm{P}} B_{\ell+1}^{\mathrm{P}}\right)$$

Only need the memory of: sn + Lsr



Boao Kong, Kun Yuan et al. *CR-Net: Scaling Parameter-Efficient Training with Cross-Layer Low-Rank Structure*, submitted to NeurIPS 2025

CR-Net: Memory saving

北京大学 PEKING UNIVERSITY

- Pre-trained on C4-en with LLaMA2-7B
- Rank: 1024 for CoLA-M, 896 for CR-Net

Table 4: Comparsion of validation perplexity (↓) and memory (↓) of different approaches in LLaMA-7B pre-training tasks. The results of compared methods are referred from [38, 57].

	Memory (GB)	10K	40K	65K	80K
8-bit Adam	72.59	N.A.	18.09	N.A.	15.47
8-bit GaLore	65.16	26.87	17.94	N.A.	15.39
Apollo	N.A.	N.A.	17.55	N.A.	14.39
CoLA-M	28.82	**22.76**	16.21	14.59	13.82
CR-Net w. re-computation	27.60	23.11	**16.01**	**14.47**	**13.72**
Training tokens (B)		1.3	5.2	8.5	10.5

- CR-Net achieves **62%** memory saving with better performance than baselines

CR-Net: Computation saving

北京大学 PEKING UNIVERSITY

Vanilla: $Y_\ell^{\mathrm{P}} = X_\ell^{\mathrm{P}} W_\ell^{\mathrm{P}}$ (smn)

(s, m) x (m, n) = (s, n) : smn

CR-Net: $Y_\ell^{\mathrm{P}} = \beta_\ell^{\mathrm{P}} Y_{\ell-1}^{\mathrm{P}} + X_\ell^{\mathrm{P}} A_\ell^{\mathrm{P}} B_\ell^{\mathrm{P}}$

(s, n) + (s, m) x (m, r) x (r, n) = sn + s(m+n)r

- s: seq. length; h: hidden dimension; L: transformer layer; s=256, batch size=1
- Rank: 512 for other baselines, 448 for CR-Net (Ensures better validation perplexity of CR-Net.)

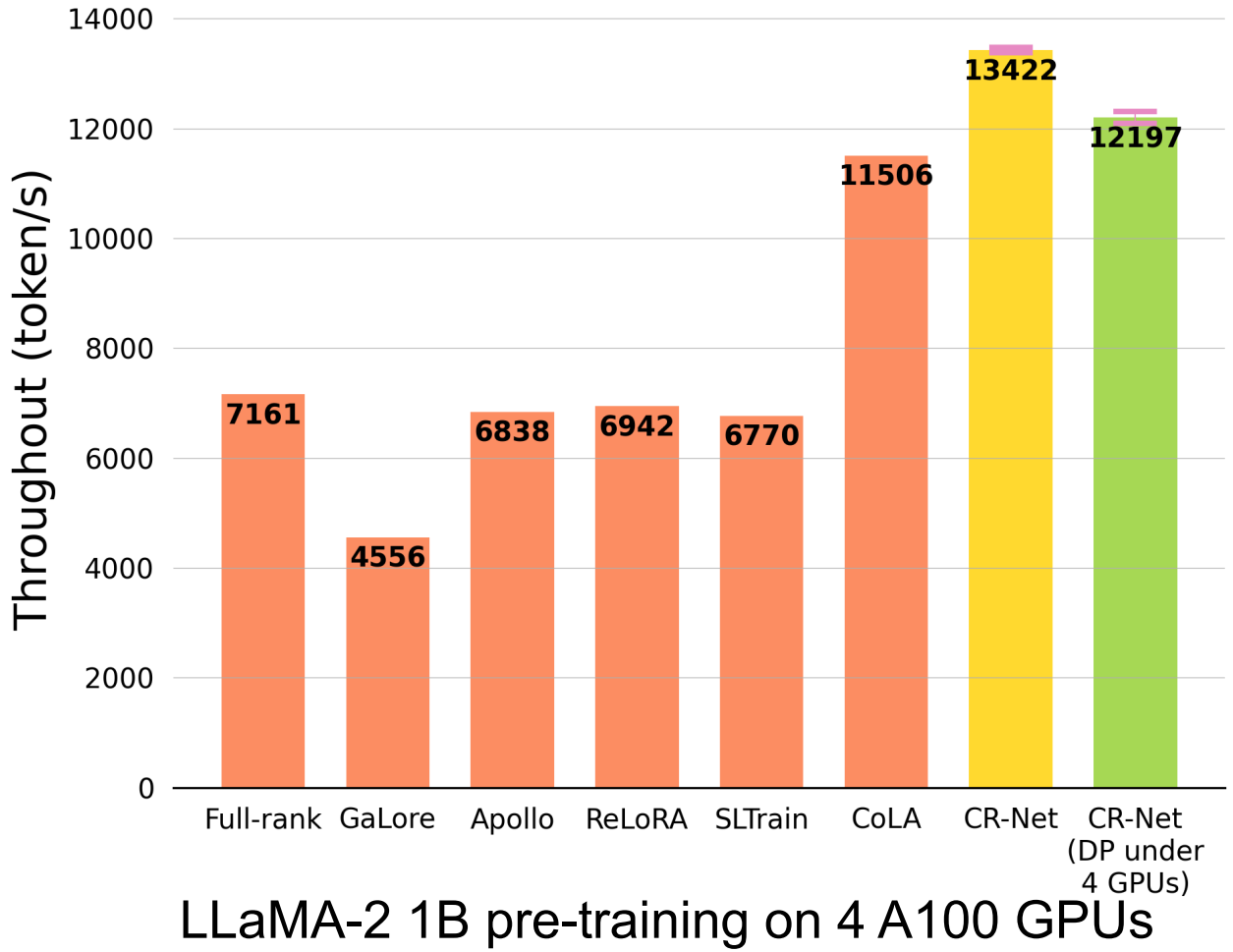
Approach	FLOPs	LLaMA-2 1B
Full-rank	$L(24sh^2 + 12s^2h + 18shh_{\mathrm{ff}})$	2.422×10^{12} (1.000×)
(Re)LoRA	$L(40sh^2 + 24s^2h + 30shh_{\mathrm{ff}})$	4.054×10^{12} (1.674×)
SLTrain	$L(24sh^2 + 12s^2h + 18shh_{\mathrm{ff}} + 24h^2r + 18hh_{\mathrm{ff}}r)$	7.164×10^{12} (2.958×)
GaLore	$L(24sh^2 + 12s^2h + 18shh_{\mathrm{ff}} + 16h^2r + 12hh_{\mathrm{ff}}r)$	5.583×10^{12} (2.305×)
CoLA	$L(48shr + 12s^2h + 18sr(h + h_{\mathrm{ff}}))$	1.005×10^{12} (0.415×)
CR-Net	$24sh^2 + 12s^2h + 18shh_{\mathrm{ff}} + (L-1)(48shr + 12s^2h + 18sr(h + h_{\mathrm{ff}}))$	0.934×10^{12} (0.385×)

- CR-Net uses **38.5%** of the FLOPs compared to the standard LLaMA-2 1B network.

CR-Net: Throughput

北京大学 PEKING UNIVERSITY

- CR-Net achieves **87%** higher throughput than the standard model.
- Even including communication overhead, CR-Net outperforms all baselines, with ~**66%** higher throughput than the standard model.



LLaMA-2 1B pre-training on 4 A100 GPUs

Hybrid between Vanilla and CR-Net

北京大学 PEKING UNIVERSITY

- Pre-training LLaMA-2 1B with sequence length s=256
- Rank: 512 for other baselines, 448 for CR-Net (Ensures better validation perplexity of CR-Net)

Algorithms	Memory (GB)	FLOPs ($\times 10^{14}$)
Vanilla GCP + Full-rank	11.98 (1.000×)	2.067 (1.000×)
CoLA-M	12.04 (1.005×)	0.764 (0.370×)
CR-Net‡	11.81(0.986×)	0.692 (0.334×)
CR-Net$^{\sharp}$	9.94(0.830×)	0.694 (0.335×)

Store full activation every 8 layers

Store full activation every 32 layers

- Compared to standard LLaMA-2 1B (vanilla GCP): At matched memory cost, CR-Net achieves **67%** faster computation.

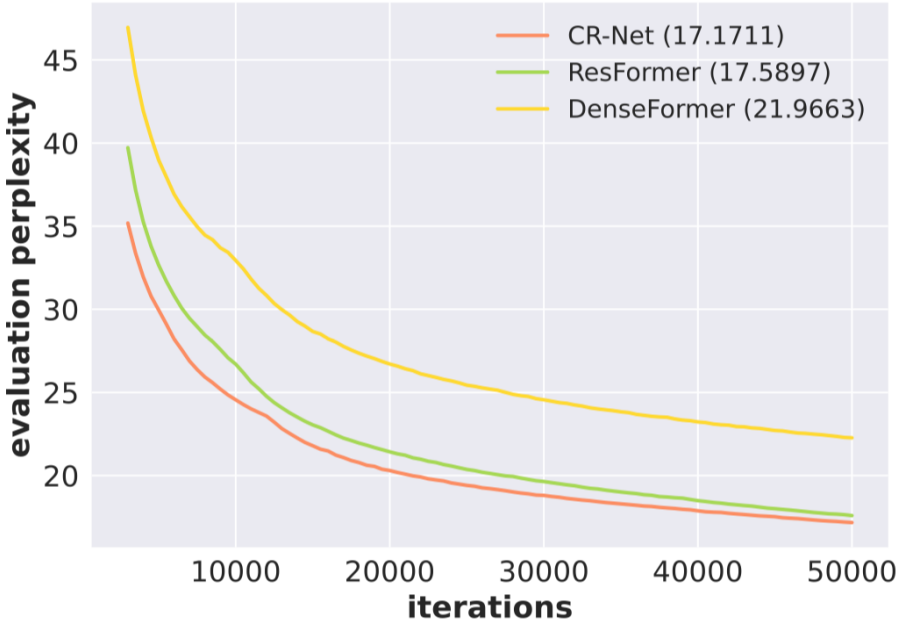
CR-Net: Efficient in both memory and compute

北京大学 PEKING UNIVERSITY

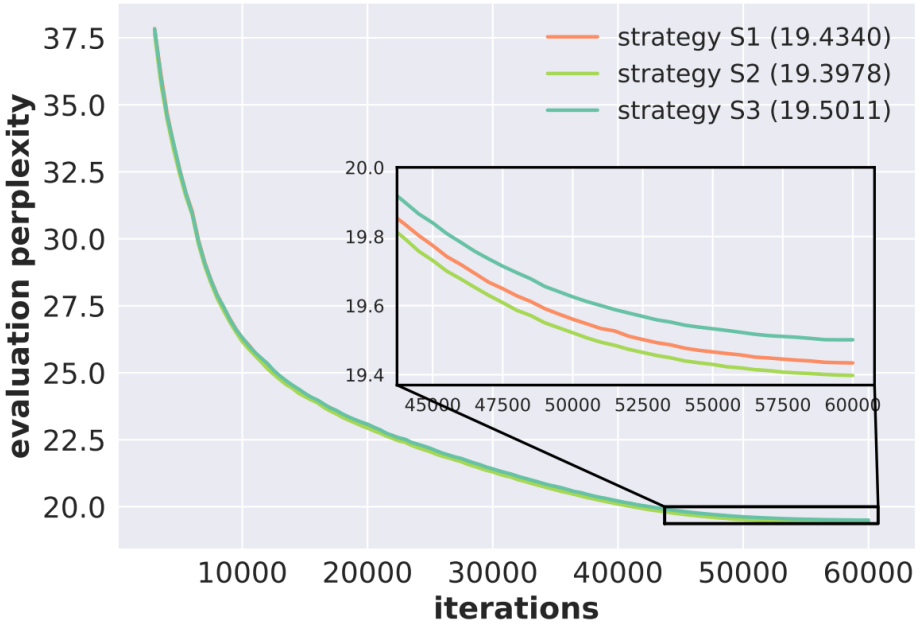
- Memory and compute complexity across methods (LLaMA-2 1B/7B, BF16)
- In CR-Net, b = number of stored full activation layers (‡: b=4, ♯: b=1)
- CoLA-M rank follows the literature; CR-Net rank r is tuned for best validation perplexity

Algorithms	LLaMA-2 1B		LLaMA-2 7B	
	Memory (GB)	FLOPs ($\times 10^{14}$)	Memory (GB)	FLOPs ($\times 10^{15}$)
Full-rank + Vanilla GCP	11.98 (1.000×)	2.133 (1.000×)	51.22 (1.000×)	2.119 (1.000×)
CoLA-M	12.04 (1.005×)	0.764 (0.358×)	24.78 (0.484×)	0.752 (0.355×)
CR-Net‡	11.81 (0.986×)	0.703 (0.330×)	23.35 (0.456×)	0.692 (0.326×)
CR-Net♯	9.94 (0.830×)	0.713 (0.334×)	22.42 (0.438×)	0.702 (0.331×)
Full-rank *w.o.* GCP	51.31 (4.283×)	0.764 (0.370×)	70.97 (1.386×)	1.608 (0.759×)

CR-Net: Ablations



Comparsion with ResFormer and DenseFormer



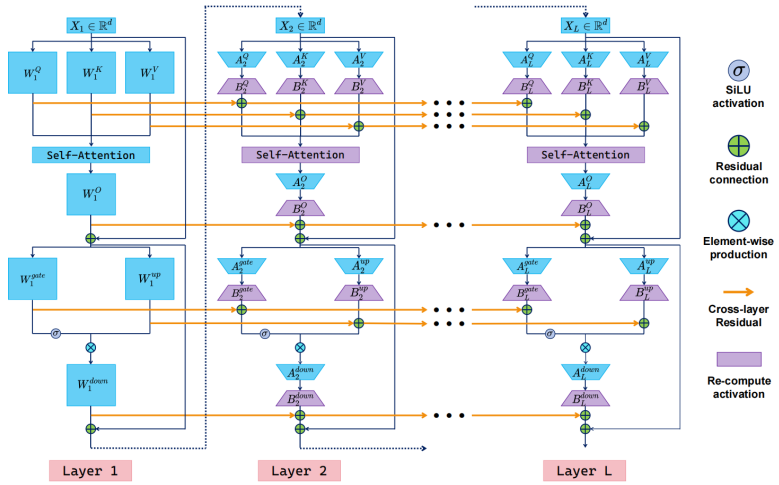
Comparsion with different low-rank strategy

R. Tian, et. al., ResFormer: Scaling ViTs with Multi-Resolution Training, CVPR 2023

Matteo Pagliardini, et. al., DenseFormer: Enhancing Information Flow in Transformers via Depth Weighted Averaging, NeurIPS 2024

Summary

- Parameter matrices are not low-rank; direct low-rank approximation leads to high error.
- **Core Idea**: Low-rank cross-layer residuals enable parameter-efficient architecture
- **Recomputation:** Tailored recomputation cuts activation memory, reducing compute by 66.6% at equal memory



- **Emprical performance**: Co-optimizes memory and compute: better performance with 43.6% fewer parameters (1B) and 38.5% less memory (7B)

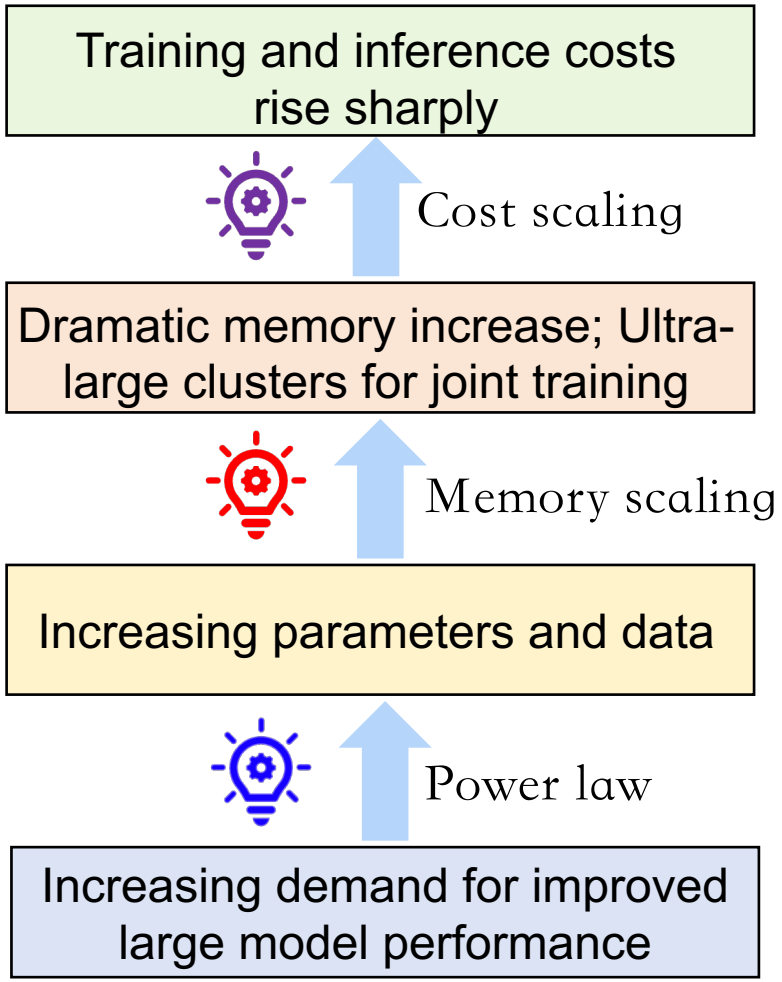
Boao Kong, Kun Yuan et al. *CR-Net: Scaling Parameter-Efficient Training with Cross-Layer Low-Rank Structure*, submitted to NeurIPS 2025

北京大学
PEKING UNIVERSITY

PART 05

Summary and Future Work

Summary



Approach 1: Design new architectures and explore novel power-law principles

Approach 2: Develop new hardware (e.g., supernodes) to reduce training and inference costs

Approach 3: Memory-efficient training methods driven by model structure

Insight: As scale increases, LLMs contain significant **structural redundancy**

Conventional training wastes memory by ignoring implicit structure

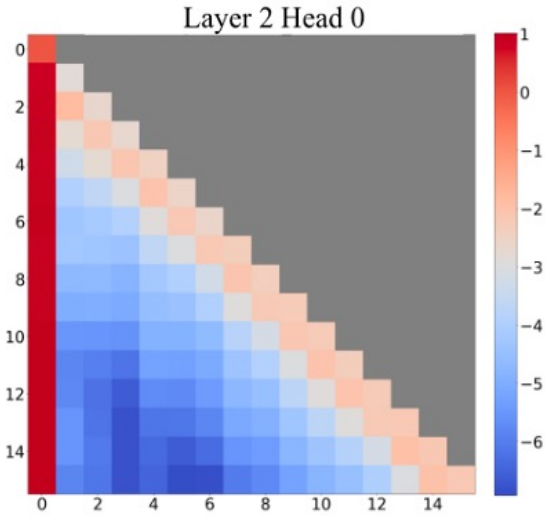
- **Low-rank gradients: Subspace training methods**
  (>75% optimizer state memory reduction)
- **Sparse activations: Importance sampling**
  (>68% FFN activation memory reduction)
- **Cross-layer low-rank activations: Parameter-efficient methods**
  (56% reduction in parameter & gradient memory)

Model structure + Hardware–software co-design = Efficient training

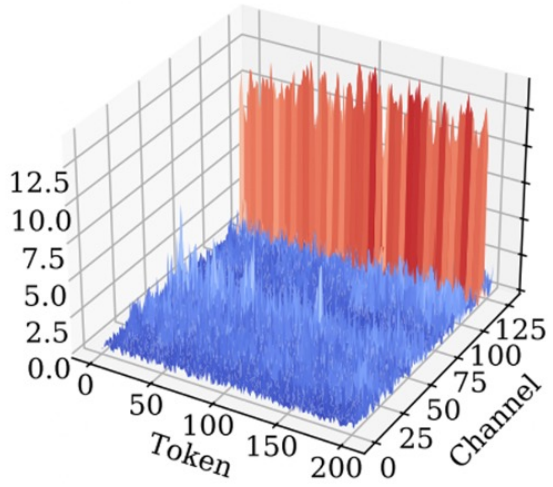
Future Work I: More Implicit Structures to Explore

Memory = **Model** + **Gradient** + **Optimizer States** + **Activations**

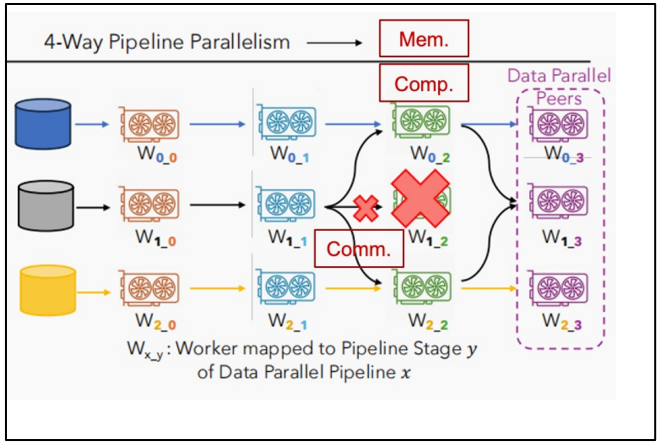
Beyond low-rank gradients and sparse/uniform activations, many hidden structures remain unexplored.



Local sparsity + Sink structures
(reduce attention memory)



Parameter distribution patterns
(improved mixed-precision strategies)



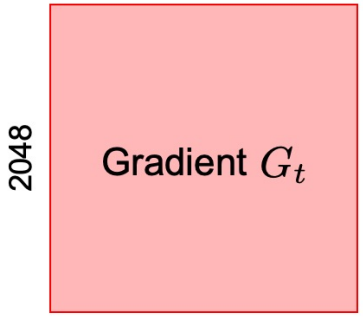
Redundancy in 3D parallel parameters
(robust training)

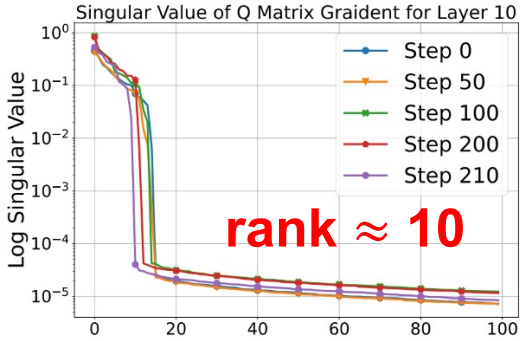
Discovering more structures will enable even more efficient training methods.

Future Work II: Saving Comm/Comp using Implicit Structures

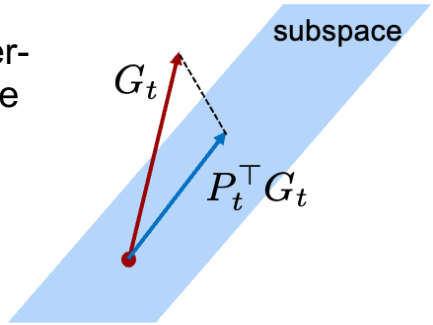
- In LLMs, compute and communication are as critical as memory.
- Similar to memory, structural properties enable reductions in communication and compute.

Low-rank gradients are projected into low-dimensional subspace for momentum updates; significant **reducing communication.**





Project into a lower-dimensional space

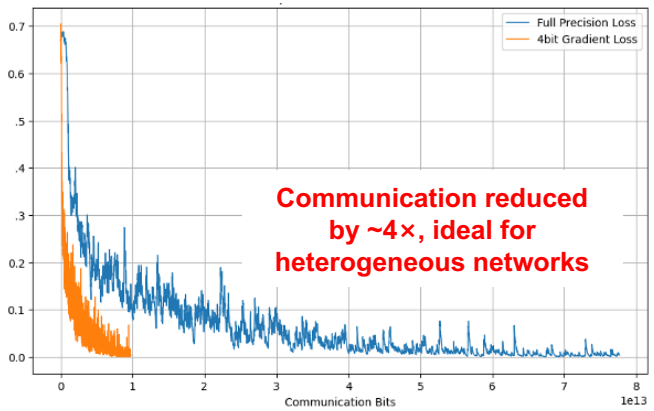


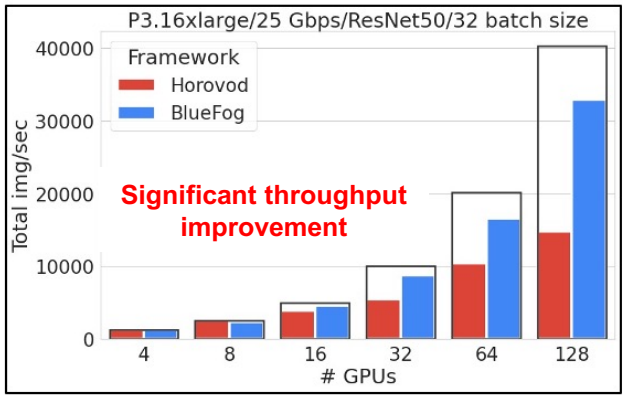
G_t (m,n) → $P_t^\top G_t$ → g_t (r,n)

$$
\begin{aligned}
\boldsymbol{g}_t &= \boldsymbol{P}_t^\top \boldsymbol{G}_t \qquad \boldsymbol{g}_t \leftarrow (1/n)\textstyle\sum_{i=1}^n \boldsymbol{g}_{t,i} \\
\boldsymbol{m}_t &= (1-\beta_1)\boldsymbol{m}_{t-1} + \beta_1 \boldsymbol{g}_t \\
\boldsymbol{v}_t &= (1-\beta_2)\boldsymbol{v}_{t-1} + \beta_2 \boldsymbol{g}_t \odot \boldsymbol{g}_t \\
\boldsymbol{\delta}_t &= \frac{\gamma}{\sqrt{\boldsymbol{v}_t}+\epsilon} \odot \boldsymbol{m}_t \\
\boldsymbol{X}_{t+1} &= \boldsymbol{X}_t - \boldsymbol{P}_t \boldsymbol{\delta}_t
\end{aligned}
$$

Low-dimensional variables use minimal communication

Low-dimensional variables use minimal memory





Chuyan Chen, Kun Yuan, et. al., Greedy Low-Rank Gradient Compression for Distributed Learning with Convergence Guarantees, arXiv: 2507.08784, 2025

Future Work II: Saving Comm/Comp using Implicit Structures

北京大学 PEKING UNIVERSITY

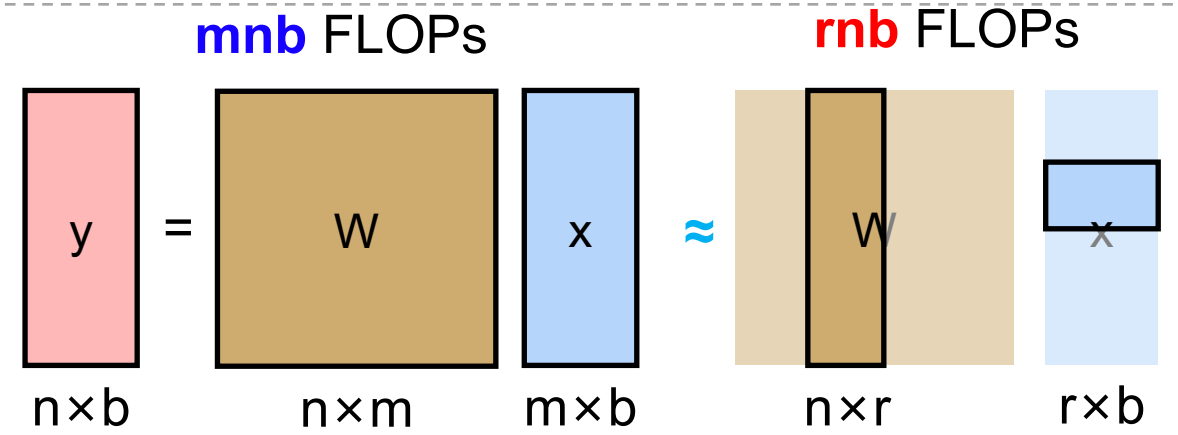
Motivation: Current memory-efficient pretraining/fine-tuning methods do not **reduce computation**

Algorithm Design: **Random matrix sampling**; Avoiding full-matrix multiplication, only important rows & columns

Forward Propagation: $y = xW_0 + xBA = x(\overline{W} + B_0A_0) + xBA$

Backward Propagation: $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\overline{W}^T + \frac{\partial L}{\partial y}(B_0A_0)^T + \frac{\partial L}{\partial y}(BA)^T$

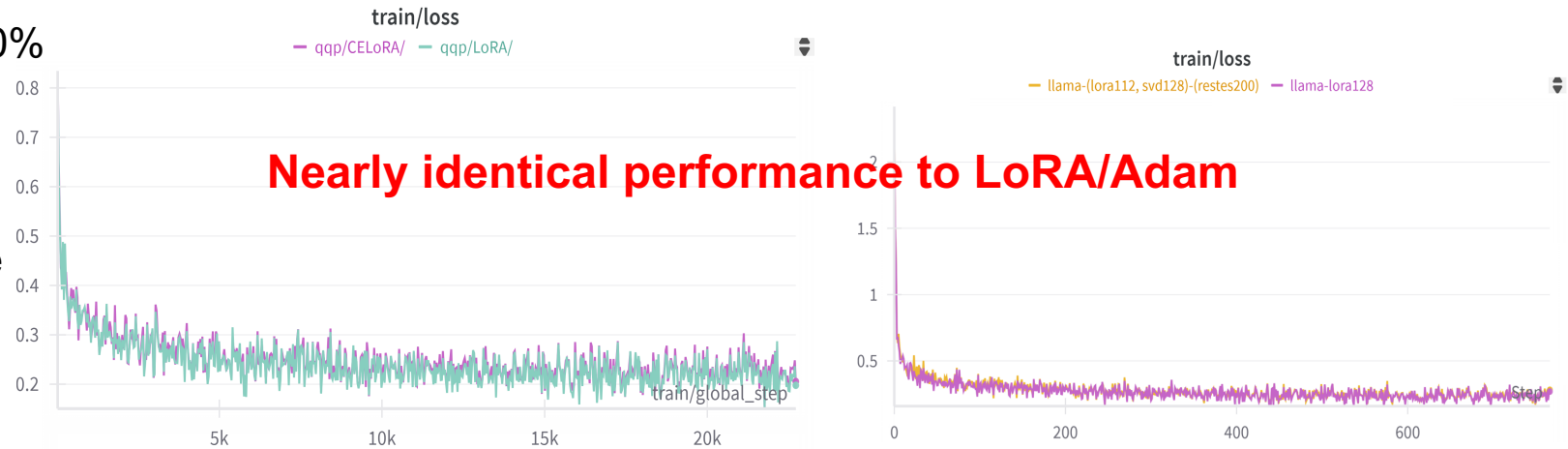
Matrix Sampling: $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}[:,i]\overline{W}^T[i,:] + \frac{\partial L}{\partial y}(B_0A_0)^T + \frac{\partial L}{\partial y}(BA)^T$



MLP sampling 90%, attention sampling 40%

Overall computation reduction: **38.57%**
Nearly lossless accuracy

Future work: During pretraining, achieve >30% computation savings with minimal accuracy loss.



G Chen, Y He, Y Hu, K Yuan, B Yuan, CE-LoRA: Computation-Efficient LoRA Fine-Tuning for Language Models, arXiv 2502.01378, 2025

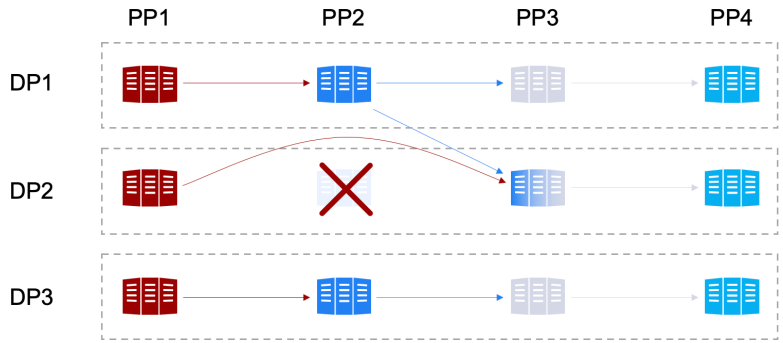
Future Work III : Robust training using Implicit Structures

北京大学 PEKING UNIVERSITY

Motivation: By 2025, clusters will scale to 100K GPUs, encompassing millions of devices. Single-device failures can halt entire training jobs, creating significant availability challenges.

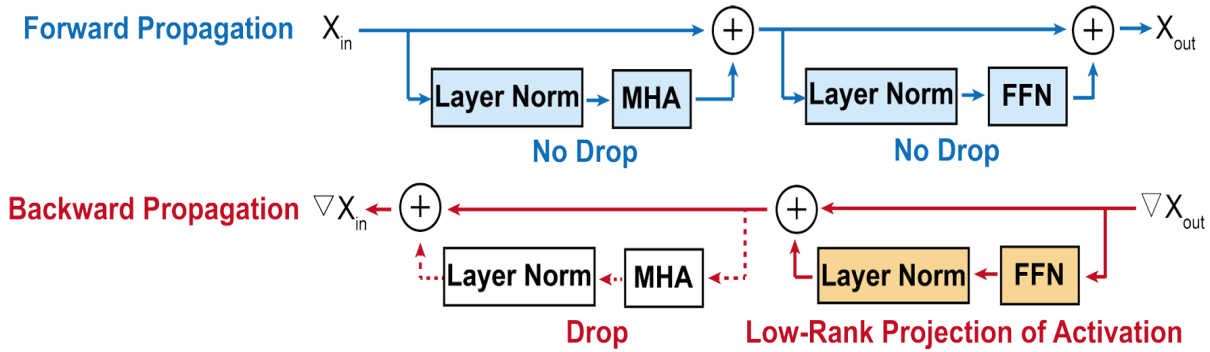
Company	Model	MTBF	Availability	100K-GPU Availability
ByteDance	175B LLM	100+ failures over 8–15 hours	~90%	~60%
Meta	Llama3.1 405B	419 failures in ~4 hours	~90%	

Proposed Solution: Efficient and elastic training to support continuous training despite GPU failures. Introduce **mixed-precision parameter redundancy**, pulling backups from neighboring devices.



Neighbor nodes run two pipeline tasks, creating memory & compute bottlenecks

Memory & Compute Trade-off: MHA activations dominate GPU memory. **Dropping the activations** can save substantial GPU memory.



R. Hu, K. Yuan, et. al., MeCeFO: Enhancing LLM Training Robustness via Fault-Tolerant Optimization, NeurIPS, 2025

Future Work III : Robust training using Implicit Structures

北京大学 PEKING UNIVERSITY

Table 2: Throughput Performance and Degradation under Different Fault Frequencies

Model	System	Throughput (tokens/s)				Throughput Drop (%)		
		No Fault	Low Freq.	Mid Freq.	High Freq.	Low Freq.	Mid Freq.	High Freq.
LLaMA-350M	Bamboo	438.06k	428.90k	421.45k	407.22k	2.09	3.79	7.04
	Oobleck	703.73k	674.15k	662.93k	632.40k	4.20	5.80	10.14
	MeCeFO	**1199.23k**	**1197.39k**	**1193.25k**	**1186.35k**	**0.15**	**0.50**	**1.07**
LLaMA-1B	Bamboo	153.75k	146.91k	144.66k	141.13k	4.45	5.91	8.21
	Oobleck	291.05k	276.05k	268.29k	250.68k	5.16	7.82	13.87
	MeCeFO	**471.19k**	**464.79k**	**461.23k**	**457.13k**	**1.36**	**2.11**	**2.98**
LLaMA-7B	Bamboo	12.41k	11.45k	10.74k	9.82k	7.73	13.42	20.84
	Oobleck	66.95k	57.05k	51.63k	48.14k	14.78	22.87	28.09
	MeCeFO	**111.12k**	**108.15k**	**107.70k**	**106.47k**	**2.67**	**3.08**	**4.18**

Validation PPL of LLaMA after MeCeFO training for the same iterations across fault frequencies

Model	No Fault	Low-frequency Fault	Medium-frequency Fault	High-frequency Fault
Llama-350M	18.74	18.75	18.88	19.04
Llama-1B	15.49	15.51	15.61	15.83
Llama-7B	14.92	14.97	15.04	15.16

R. Hu, K. Yuan, et. al., MeCeFO: Enhancing LLM Training Robustness via Fault-Tolerant Optimization, Submitted to NeurIPS 2025, 2025

Future work IV: Power-Law Optimized Model Architectures

北京大学 PEKING UNIVERSITY

Training and inference costs rise sharply

Cost scaling

Dramatic memory increase; Ultra-large clusters for joint training

Memory scaling

Increasing parameters and data

Power law

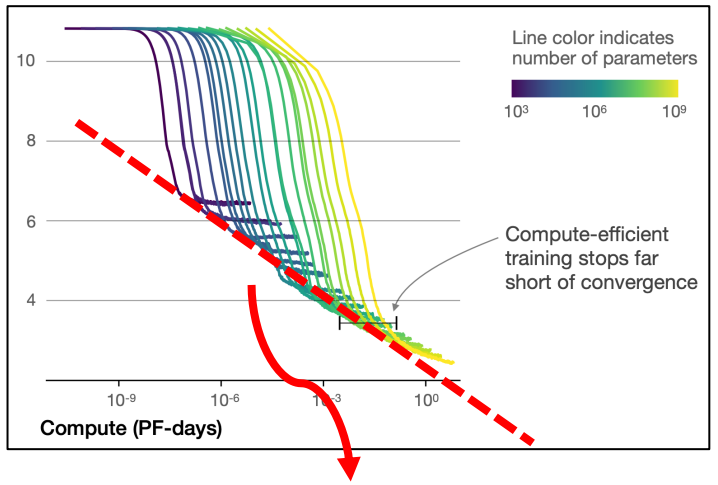
Increasing demand for improved large model performance

Approach 1: Design new architectures and explore novel power-law principles

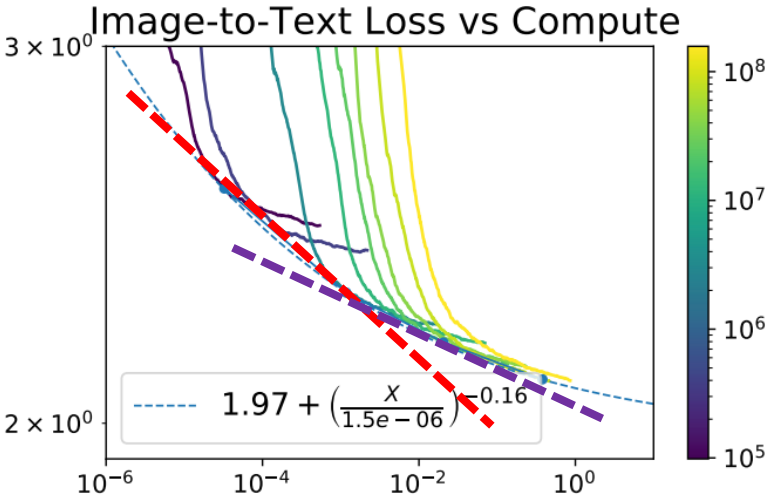
Approach 2: Develop new hardware (e.g., supernodes) to reduce training and inference costs

Approach 3: Memory-efficient training methods driven by model structure

Improving scaling Laws is a more fundamental direction to efficient training



Transformer performance ceiling is hard to surpass



Scaling slope declines

LLM Architecture: Linear Attention

北京大学 PEKING UNIVERSITY

Transformer Model

- ✔ Efficient Training: support parallelism
- ✘ Inefficient Inference: KV cache requires $O(s)$ memory

RNN Model

- ✘ Inefficient training: no parallelism
- ✔ Efficient Inference: $O(1)$ memory

Transformer Training Paradigm

How to design more efficient linear attention?

RNN Inference Paradigm

Exp Activation

$\exp(QK^\top)$

Kernel Activation

$\phi(Q)\phi(K)^\top$

Linear Attention[1]

- ✔ Efficient Training: match Transformers
- ✔ Efficient Inference: convert to RNN without loss

$$\frac{\sum \phi(q)\phi(k_i)^\top v_i}{\sum \phi(q)\phi(k_i)^\top}$$

$$\frac{\phi(q)[\sum \phi(k_i)^\top v_i]}{\phi(q)[\sum \phi(k_i)^\top]}$$

Training Paradigm

Inference Paradigm

References

[NeurIPS 2025] R. Hu, Y. He, R. Yan, M. Sun, B. Yuan, **K. Yuan***, “MeCeFO: Enhancing LLM Training Robustness via Fault-Tolerant Optimization”, Advances in Neural Information Processing Systems (NeurIPS), 2025.

[ICML 2025] Y. He, P. Li, Y. Hu, C. Chen, and **K. Yuan***, “Subspace Optimization for Large Language Models with Convergence Guarantees”, International Conference on Machine Learning (ICML), 2025.

[ICML 2025] Y. Chen, Y. Zhang, Y. Liu, **K. Yuan***, and Z. Wen, “A Memory Efficient Randomized Subspace Optimization Method for Training Large Language Models”, International Conference on Machine Learning (ICML), 2025.

[ICML 2025] Y. Song, P. Li, B. Gao, and **K. Yuan***, “Distributed Retraction-Free and Communication-Efficient Optimization on the Stiefel Manifold”, International Conference on Machine Learning (ICML), 2025.

[ICML 2025] L. Liang, G. Luo, X. Chen, and **K. Yuan***, “Achieving Linear Speedup and Optimal Complexity for Decentralized Optimization over Row-stochastic Networks”, International Conference on Machine Learning (ICML), 2025.

[ICML 2025] L. Chen, Q. Xiao, E. H. Fukuda, X. Chen, **K. Yuan**, and T. Chen, “Efficient Multi-Objective Learning under Preference Guidance: A First-Order Penalty Approach”, International Conference on Machine Learning (ICML), 2025.

[ICLR 2025] Y. Chen, Y. Zhang, L. Cao, **K. Yuan***, and Z. Wen, “Enhancing Zeroth-Order Fine-Tuning for Language Models with Low-Rank Structures”, International Conference on Learning Representations (ICLR), 2025.

[NeurIPS 2024] S. Zhu, B. Kong, S. Lu, X. Huang, and **K. Yuan***, “SPARKLE: A Unified Single-Loop Primal- Dual Framework for Decentralized Bilevel Optimization”, Advances in Neural Information Processing Systems (NeurIPS), 2024

[ICML 2024] Y. He, J. Hu, X. Huang, S. Lu, B. Wang, and **K. Yuan***, “Distributed Bilevel Optimization with Communication Compression”, International Conference on Machine Learning (ICML), 2024.

References

[NeurIPS 2023] Y. He, X. Huang, and **K. Yuan***. “Unbiased Compression Saves Communication in Distributed Optimization: When and How Much?”, Advances in Neural Information Processing Systems (NeurIPS), 2023.

[ICML 2023] L. Ding, K. Jin, B. Ying, **K. Yuan**, and W. Yin. “DSGD-CECA: Decentralized SGD with Communication-Optimal Exact Consensus Algorithm”, The International Conference on Machine Learning (ICML), 2023.

[NeurIPS 2022] X. Huang, Y. Chen, W. Yin, and **K. Yuan***, “Lower Bounds and Nearly Optimal Algorithms in Distributed Learning with Communication Compression”, Neural Information Processing Systems (NeurIPS), 2022.

[NeurIPS 2022] Z. Song, W. Li, K. Jin, L. Shi, M. Yan, W. Yin, and **K. Yuan***, “Communication-Efficient Topologies for Decentralized Learning with O(1) Consensus Rate”, Neural Information Processing Systems (NeurIPS), 2022.

[NeurIPS 2022] **K. Yuan*,** X. Huang, Y. Chen, X. Zhang, Y. Zhang, and P. Pan, “Revisiting Optimal Convergence Rate for Smooth and Non-Convex Stochastic Decentralized Optimization”, Neural Information Processing Systems (NeurIPS), 2022

Preprint and New Submissions:

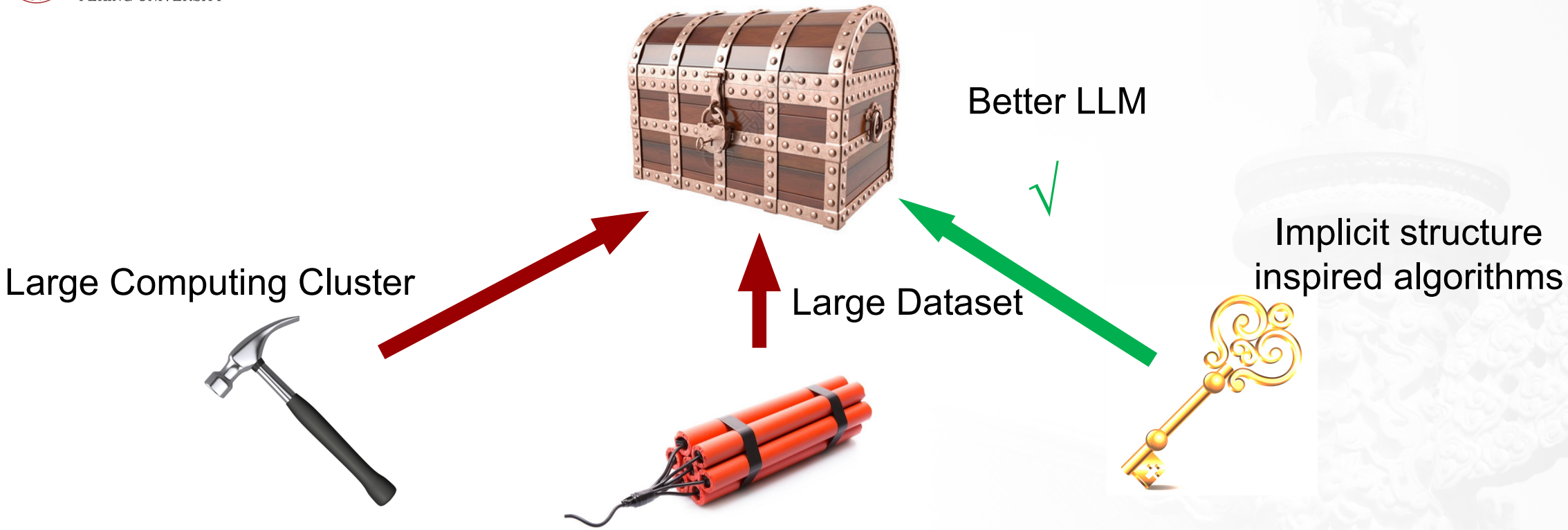
T. Wu, Y. He, B. Wang, and **K. Yuan***, *Mixture-of-Channels: Exploiting Sparse FFNs for Efficient LLMs Pre-Training and Inference, 2025*

B. Kong, J. Liang, Y. Liu, R. Deng, and **K. Yuan***, *CR-Net: Scaling Parameter-Efficient Training with Cross-Layer Low-Rank Structure, 2025*

B. Kong, X. Huang, Y. Xu, Y. Liang, B. Wang, and **K. Yuan***, *Clapping: Removing Per-sample Storage for Pipeline Parallel Learning with Communication Compression, 2025*

C Chen, Y He, P Li, W Jia, **K Yuan***, Greedy Low-Rank Gradient Compression for Distributed Learning with Convergence Guarantees, arXiv:2507.08784, 2025

北京大学
PEKING UNIVERSITY



Thanks!

Kun Yuan homepage: https://kunyuan827.github.io/