北京大学
PEKING UNIVERSITY

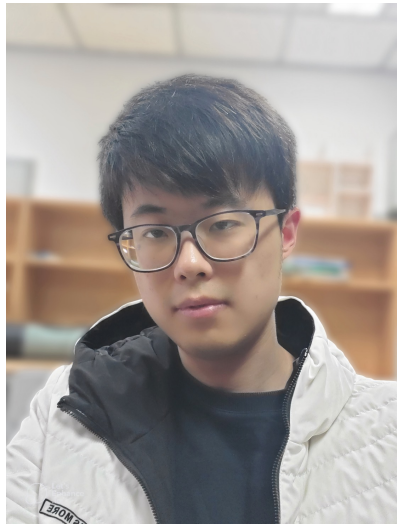
A Memory Efficient Subspace Optimization Method for Training Large Language Models

Kun Yuan (袁 坤)

Center for Machine Learning Research @ Peking University

June 24, 2025 Beijing

Joint work with



Yiming Chen
(PKU)

Yuan Zhang
(PKU)

Yin Liu
(PKU)

Zaiwen Wen
(PKU)

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

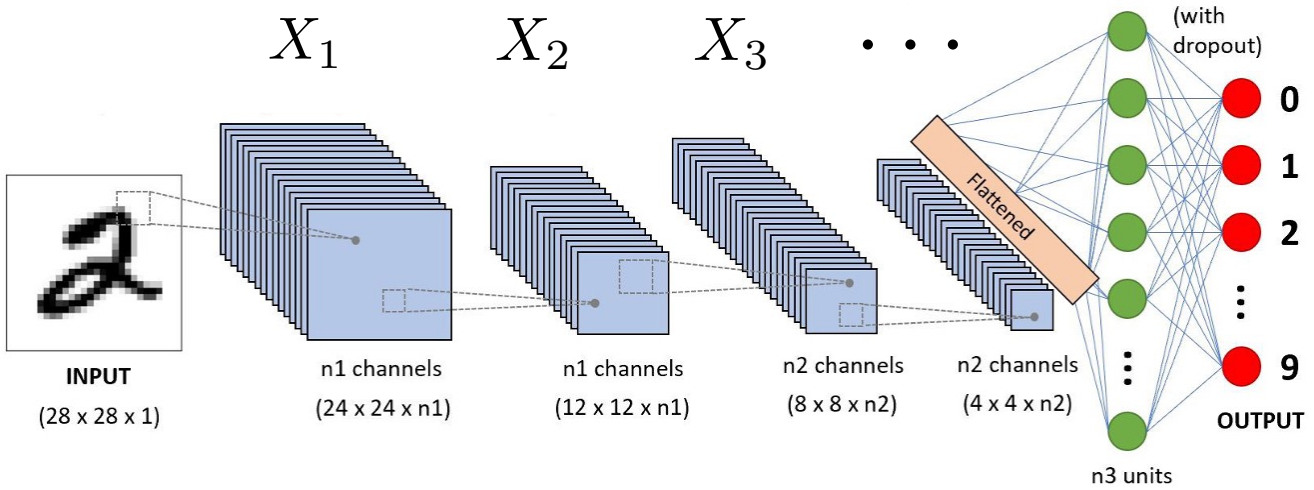
北京大学
PEKING UNIVERSITY

PART 01

Basics and Motivation

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

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

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

Optimizer states introduces significant memory cost

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 $z_1, z_2, \cdots, z_L$

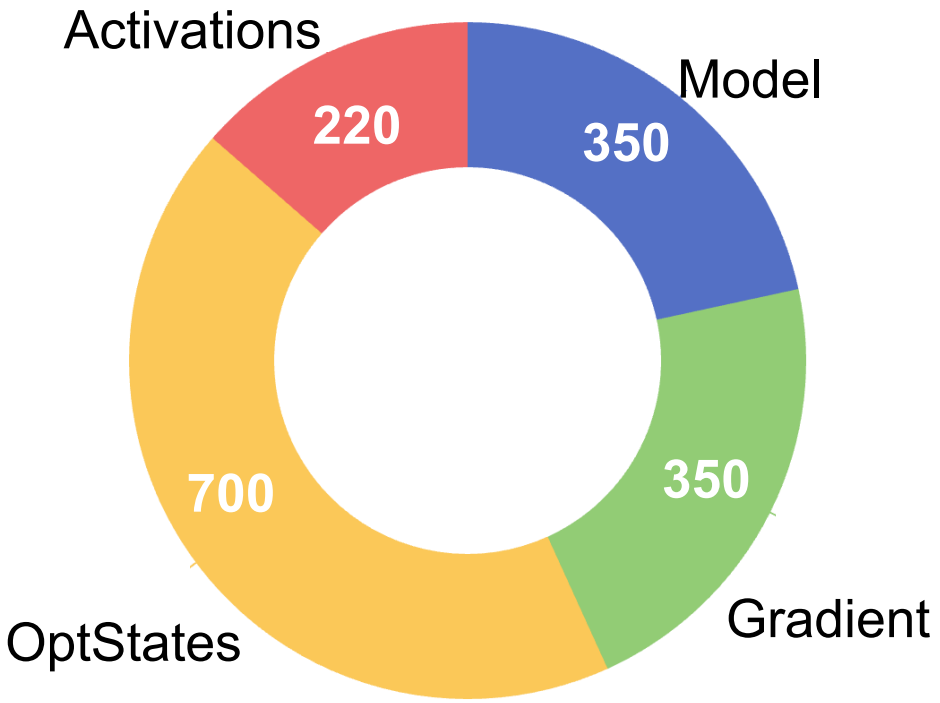
- The size of activations depends on sequence length and batch size

Minimum memory requirement: GPT-3

北京大学 PEKING UNIVERSITY

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

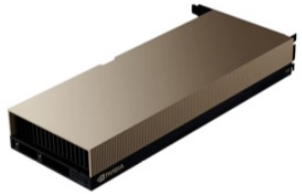
  - Parameters: 175B
  - Model storage: 175B * 2 Bytes = 350 GB
  - Gradient storage: 350 GB
  - Optimizer states: 700 GB (using Adam)
  - Activation storage: ~220 GB
  - In total: **1620 GB**



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**



A100 80G

x 21

The minimum requirement is A100 * 21

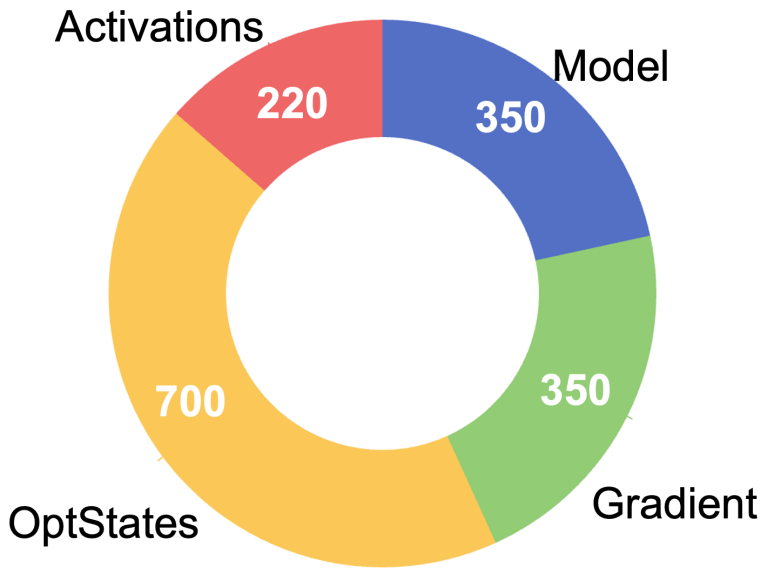
Very expensive!

Memory-efficient algorithm is in urgent need

北京大学 PEKING UNIVERSITY

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

- With memory-efficient algorithms, we can
  - **Train larger models** on limited computing resources
  - Use a larger training batch size to **improve throughput**
- The memory for model cannot be saved
- **Is it possible to save memory for gradient, optimizer states, and activations?**



北京大学
PEKING UNIVERSITY

PART 02

Randomized Subspace Optimization

Randomized Subspace Optimization (RSO)

- Consider the optimization problem for the LLM training:

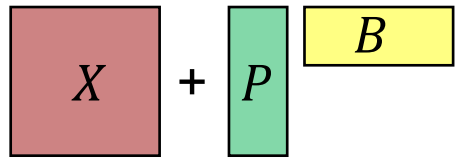
$$\min_{\boldsymbol{X} \in \mathbb{R}^{m \times n}} f(\boldsymbol{X}) := \mathbb{E}_{\boldsymbol{\xi}}[F(\boldsymbol{X}; \boldsymbol{\xi})].$$

- We solve small problems sequentially; optimizing over **one subspace** (spanned by P) at a time:

Sample P for each iteration k

$$\tilde{\boldsymbol{B}}^k \approx \operatorname*{argmin}_{\boldsymbol{B} \in \mathbb{R}^{r \times n}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k \boldsymbol{B}) + \frac{1}{2\eta^k} \|\boldsymbol{B}\|^2 \right\},$$

$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k \tilde{\boldsymbol{B}}^k.$$



Here $X \in R^{m \times n}$, $P \in R^{m \times r}$ and $B \in R^{r \times n}$. Dimension $r \ll \min\{m, n\}$. The new variable B has much fewer parameters than X

Randomized Subspace Optimization (RSO)

北京大学 PEKING UNIVERSITY

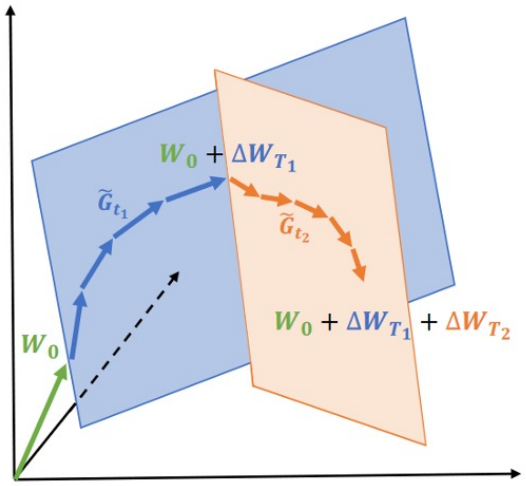
Can be solved with any optimizer

Promote strong convexity

$$\tilde{\boldsymbol{B}}^k \approx \underset{\boldsymbol{B}\in\mathbb{R}^{r\times n}}{\operatorname{argmin}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k \boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2 \right\}, \tag{4a}$$

$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k \tilde{\boldsymbol{B}}^k. \tag{4b}$$

(Zhao et. al. ICML 2023)



Algorithm 1: Randomized Subspace Optimization

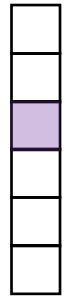
Input: Initialization $\boldsymbol{X}^0$
Output: Solution $\boldsymbol{X}^K$
for $k = 0, 1, \ldots, K-1$ **do**
  Sample $\boldsymbol{P}^k$ according to a given distribution;
  Solve subproblem (4a) and obtain the approximate solution $\tilde{\boldsymbol{B}}^k$ using a given optimizer such as Adam;
  Update the weights by $\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k \tilde{\boldsymbol{B}}^k$;

RSO is essentially a generalized method of coordinate minimization

- Recall the random **coordinate** minimization:

$$b_k^\star = \arg\min_{b\in\mathbb{R}} \{f(x^k + b\cdot e_{i_k})\}, \text{ where } i_k \sim \mathcal{U}\{1,\dots,d\},$$
$$x^{k+1} = x^k + b_k^\star \cdot e_{i_k},$$

b is the increment in the i-th coordinate

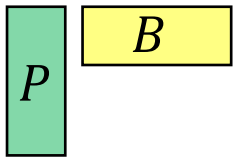


where the variable x is a vector in the above method

- Random subspace optimization can be regarded as a generalized method; not in the coordinate subspace but in the low-rank subspace

$$\tilde{\boldsymbol{B}}^k \approx \operatorname*{argmin}_{\boldsymbol{B}\in\mathbb{R}^{r\times n}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2 \right\},$$
$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k\tilde{\boldsymbol{B}}^k.$$

B is the increment in the subspace spanned by P^k



RSO saves gradient and optimizer state

北京大学 PEKING UNIVERSITY

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

$$\tilde{\boldsymbol{B}}^k \approx \underset{\boldsymbol{B}\in\mathbb{R}^{r\times n}}{\operatorname{argmin}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2 \right\},$$
$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k\tilde{\boldsymbol{B}}^k.$$

Let $\boldsymbol{h}(\boldsymbol{B}) = f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta_k}\|\boldsymbol{B}\|_F^2$

We solve $\tilde{\boldsymbol{B}}^k \approx \arg\min_{\boldsymbol{B}}\{h(\boldsymbol{B})\}$ using Adam

For $t = 1, 2, \cdots, \tau$ — Adam for solving $h(B)$

$$\boldsymbol{g}_t = \nabla_{\boldsymbol{B}} h(\boldsymbol{B}_t)$$
$$\boldsymbol{m}_t = (1-\beta_1)\boldsymbol{m}_{t-1} + \beta_1\boldsymbol{g}_t$$
$$\boldsymbol{v}_t = (1-\beta_2)\boldsymbol{v}_{t-1} + \beta_2\boldsymbol{g}_t \odot \boldsymbol{g}_t$$
$$\boldsymbol{B}_{t+1} = \boldsymbol{B}_t - \frac{\gamma}{\sqrt{\boldsymbol{v}_t + \epsilon}} \odot \boldsymbol{m}_t$$

RSO saves gradient and optimizer state

$$h(B) = f(X^k + P^k B) + \frac{1}{2\eta_k}\|B\|_F^2$$

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

Adam for solving $f(X)$

$$G_t = \nabla F(X_t; \xi_t) \quad \text{(m, n)}$$
$$M_t = (1-\beta_1)M_{t-1} + \beta_1 G_t \quad \text{(m, n)}$$
$$V_t = (1-\beta_2)V_{t-1} + \beta_2 G_t \odot G_t \quad \text{(m, n)}$$
$$X_{t+1} = X_t - \frac{\gamma}{\sqrt{V_t + \epsilon}} \odot M_t \quad \text{(m, n)}$$

Adam for solving $h(B)$

$$g_t = \nabla_B h(B_t) \quad \text{(r, n)}$$
$$m_t = (1-\beta_1)m_{t-1} + \beta_1 g_t \quad \text{(r, n)}$$
$$v_t = (1-\beta_2)v_{t-1} + \beta_2 g_t \odot g_t \quad \text{(r, n)}$$
$$B_{t+1} = B_t - \frac{\gamma}{\sqrt{v_t + \epsilon}} \odot m_t \quad \text{(r, n)}$$

- Reduce gradient and optimizer states from mn parameters to rn parameters;
- Incurs **tiny** gradient and optimizer when $r \ll m$

RSO saves activation memory

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

Forward: $Z = YX,\ y = L(Z).$

Backward: $\frac{\partial y}{\partial X} = Y^\top \frac{\partial y}{\partial Z}$

Adam stores (s by m) activations

Forward: $Z = Y(X + PB),\ y = L(Z).$

Backward: $\frac{\partial y}{\partial B} = (YP)^\top \frac{\partial y}{\partial Z}$

RSO stores (s by r) activations

- Reduce activations from sm parameters to sr parameters;
- Incurs **tiny** activations when $r \ll m$

Memory comparison with existing optimizer

s: #tokens in a sequence n: hidden dimension b: batch size r: rank

Algorithm	Memory		
	Optimizer States	Activations	**Gradients**
RSO	$24nr$	$8bsn + 4bsr + 2bs^2$	$12nr$
GaLore	$24nr$	$15bsn + 2bs^2$	$12n^2$
LoRA	$48nr$	$15bsn + 2bs^2$	$24nr$
Adam	$24n^2$	$15bsn + 2bs^2$	$12n^2$

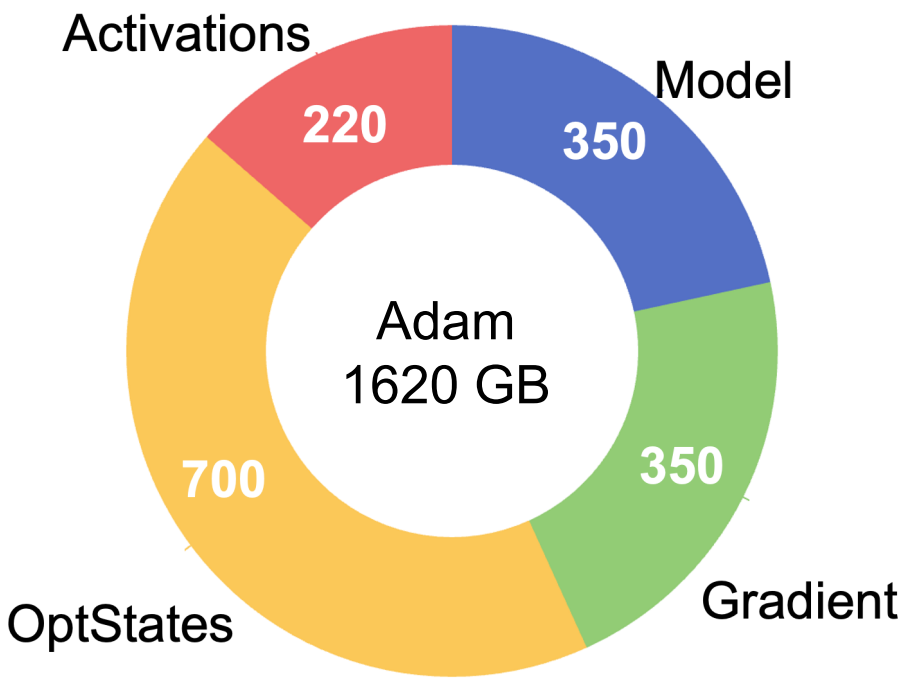
$$\tilde{\boldsymbol{B}}^k \approx \underset{\boldsymbol{B}\in\mathbb{R}^{r\times n}}{\operatorname{argmin}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2 \right\},$$

$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k\tilde{\boldsymbol{B}}^k.$$

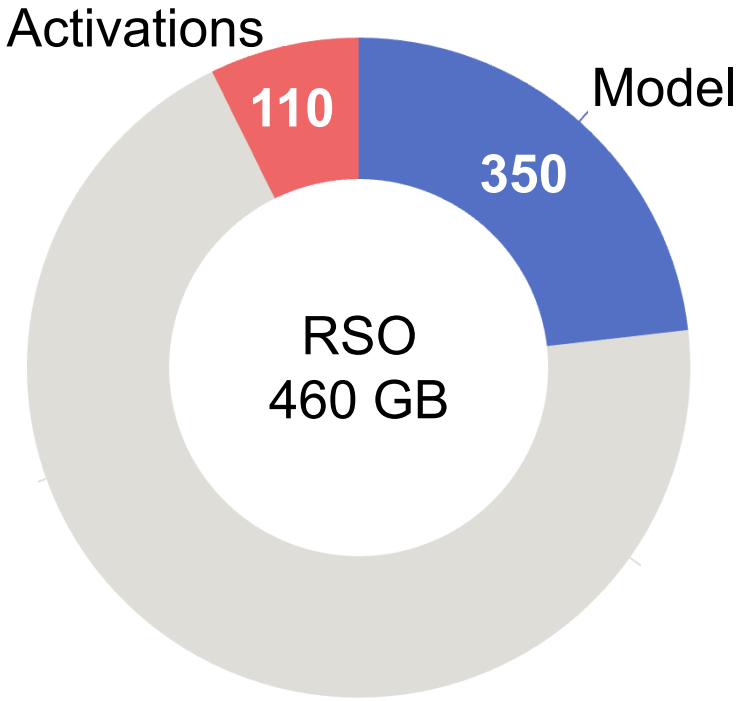
- Saves gradient, optimizer states, and activations simultaneously
- RSO is the most memory-efficient optimizer to our best knowledge

Memory saving visualization

Suppose $r \ll m$, and s^2 is not the dominant term



21 x A100



6 x A100

北京大学
PEKING UNIVERSITY

PART 03

Convergence Guarantees

Assumptions

Definition 5.1 (Expected ϵ-inexact solution). *A solution $\tilde{\boldsymbol{B}}^k$ is said to be an expected ϵ-inexact solution if it satisfies:*

$$\mathbb{E}[g^k(\tilde{\boldsymbol{B}}^k)] - g^k(\boldsymbol{B}_\star^k) \leq \epsilon,$$

where $g^k(\boldsymbol{B}) := f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2$, and $\boldsymbol{B}_\star^k$ is the optimal solution define as $\boldsymbol{B}_\star^k := \arg\min_{\boldsymbol{B}} g^k(\boldsymbol{B})$.

Assumption 5.2. *The objective function $f(\boldsymbol{W})$ is L-smooth, i.e., it holds for any $\boldsymbol{W}^1$ and $\boldsymbol{W}^2$ that*

$$\|\nabla f(\boldsymbol{W}^1) - \nabla f(\boldsymbol{W}^2)\| \leq L\|\boldsymbol{W}^1 - \boldsymbol{W}^2\|,$$

(smoothness)

where $\|\boldsymbol{W}\| := \sqrt{\sum_{\ell=1}^{\mathcal{L}} \|W_\ell\|_F^2}$ for any $\boldsymbol{W} = \{W_\ell\}_{\ell=1}^{\mathcal{L}}$.

Assumption 5.3. *The random matrix $\boldsymbol{P} = \{P_\ell\}_{\ell=1}^{\mathcal{L}}$ is sampled from a distribution such that $P_\ell^\top P_\ell = (m_\ell/r_\ell)I_{r_\ell}$ and $\mathbb{E}[P_\ell P_\ell^\top] = I_{m_\ell}$ for each ℓ.*

(random subspace)

RSO convergence theorem

北京大学 PEKING UNIVERSITY

Theorem

Let each subproblem in RSO iteration be solved starting from the initial point $\boldsymbol{B}^0 = \mathbf{0}$ to an expected ϵ-inexact solution $\tilde{\boldsymbol{B}}^k$ with a suitable choice of η^k. The sequence $\{\boldsymbol{X}^k\}$ generated by the RSO method satisfies the following bound:

subproblem optimizer

$$\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\|\nabla f(\boldsymbol{X}^k)\|^2 \leq \boxed{\frac{18\hat{L}\Delta_0}{K}} + \boxed{18\hat{L}\epsilon},$$

RSO framework

where $\Delta_0 := f(\boldsymbol{X}^0) - f^*$ and $\hat{L} := \max_\ell\{m_\ell/r_\ell\}L$.

- RSO converges to ϵ-accurate solution at rate $1/K$
- The accuracy is determined by the subproblem optimizer

RSO convergence theorem with concrete subproblem optimizer

$$h(B) = f(X^k + P^k B) + \frac{1}{2\eta_k}\|B\|_F^2$$

Subproblem Solver	Subproblem Complexity	Total Complexity
Stochastic ZO Method	$O((\sum_{\ell=1}^{\mathcal{L}} n_\ell r_\ell)^2 \epsilon^{-2})$	$O((\sum_{\ell=1}^{\mathcal{L}} n_\ell r_\ell)^2 \epsilon^{-3})$
GD	$O(\log \epsilon^{-1})$	$\tilde{O}(\epsilon^{-1})$
Accelerated GD	$O(\log \epsilon^{-1})$	$\tilde{O}(\epsilon^{-1})$
SGD	$O(\epsilon^{-1})$	$O(\epsilon^{-2})$
Momentum SGD	$O(\epsilon^{-1})$	$O(\epsilon^{-2})$
Adam-family	$O(\epsilon^{-1})$	$O(\epsilon^{-2})$
Newton's method	$O(\log(\log \epsilon^{-1}))$	$\tilde{O}(\epsilon^{-1})$
Stochastic Quasi-Newton method	$O(\epsilon^{-1})$	$O(\epsilon^{-2})$

It is observed that RSO with Adam to solve subproblem has sample complexity $O(\epsilon^{-2})$, which is on the same order as vanilla Adam without subspace projection.

北京大学
PEKING UNIVERSITY

PART 04

Experiments

RSO performance： Memory comparison in Pre-train

北京大学 PEKING UNIVERSITY

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	**34.55**(0.14G)	**25.34** (0.27G)	**18.86** (0.49G)	15.86 (1.46G)
r/d_{model}	128 / 256	256 / 768	256 / 1024	512 / 2048
Training Tokens (B)	1.1	2.2	6.4	13.1

Table: Comparison of validation perplexity and estimated memory usage for optimizer states across different algorithms during the pre-training of LLaMA models of various sizes on the C4 dataset.

- Saves significant optimizer states memory； performance degradation is within 2%
- Run for the same number of iterations/samples; performs well for small models (<350M, ~1%)
- Performance deteriorates for relatively large models ($\geq$ 1B, 2%)

RSO performance：Memory comparison in Pre-train

Practical **Total Memory** Profiling (not just optimizer)

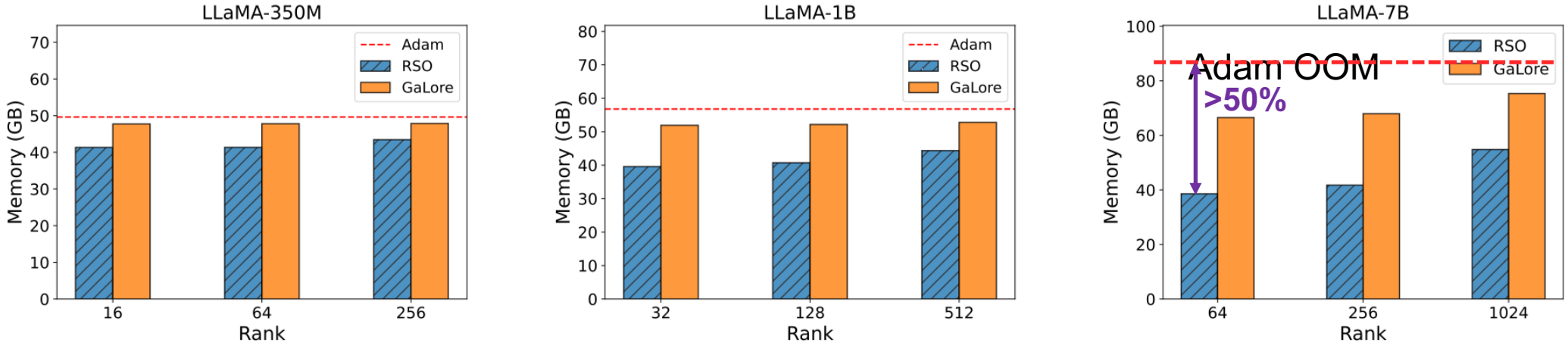


Figure 2: Comparison of peak memory usage (in GB) per device for RSO and GaLore during LLaMA training with varying ranks. All hyperparameters, except rank, are consistent with [Zhao et al., 2024a]. Adam's memory usage is reported for LLaMA-350M and LLaMA-1B but excluded for LLaMA-7B due to an out-of-memory (OOM) error.

- RSO saves gradient, optimizer, and activations, and hence it is more memory-efficient
- As rank increases, memory saving in RSO gets decreased

RSO performance：Computation comparison in Pre-train

Method	LLaMA-1B (Seconds)			LLaMA-7B (Seconds)		
	Seq 64	Seq 128	Seq 256	Seq 64	Seq 128	Seq 256
RSO	0.94	1.70	3.29	2.40	2.94	4.60
GaLore	1.12	1.84	3.35	7.86	8.26	9.12
Adam	1.11	1.81	3.32	7.84	8.23	OOM

Table 5: Comparison of iteration time (in seconds) for different methods in LLaMA training across various sequence lengths. All hyperparameters, except sequence length, follow [Zhao et al., 2024a]. LLaMA-1B runs on 4× A800 GPUs, while LLaMA-7B uses 8× A800 GPUs. SVD decomposition time in GaLore is excluded. Additionally, for LLaMA-7B with a sequence length of 256, the Adam optimizer encounters an out-of-memory (OOM) error.

- RSO achieves much faster training speed due to efficient solving for small subproblems

RSO performance：end-to-end pre-train

北京大学 PEKING UNIVERSITY

Run for the same number of iterations

Method	Memory (GB)	Training Time (h)	Perplexity
Adam	78.92	216	15.43
GaLore	75.33	134	15.59
RSO	54.81	64	15.99

Table 6: Comparison of various pre-training methods for the LLaMA-7B model on the C4 dataset. Perplexity is reported at 50K steps. The training is conducted on 8× A800 GPUs. The actual memory cost per device and the total training time are also reported. RSO and GaLore are configured with a batch size of 16, while Adam uses a batch size of 8.

- With ~3.6% performance degradation, RSO achieves **3.7x** speedup and **30%** memory saving
- It is conjectured that, given the same runtime, RSO outperforms Adam; however, we have not conducted experiments to verify this.

Summary

北京大学 PEKING UNIVERSITY

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

- Random subspace optimization (RSO) saves **gradient, optimizer, and activations**

$$\tilde{\boldsymbol{B}}^k \approx \underset{\boldsymbol{B}\in\mathbb{R}^{r\times n}}{\operatorname{argmin}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2 \right\},$$
$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k\tilde{\boldsymbol{B}}^k.$$

- RSO has strong convergence guarantees
- **Open questions**: How to improve RSO performance, especially for large LLMs?

paper

Github Code

Ongoing work: Explore better subspace than random

$$\tilde{\boldsymbol{B}}^k \approx \underset{\boldsymbol{B}\in\mathbb{R}^{r\times n}}{\operatorname{argmin}} \left\{ f(\boldsymbol{X}^k + \boldsymbol{P}^k\boldsymbol{B}) + \frac{1}{2\eta^k}\|\boldsymbol{B}\|^2 \right\},$$
$$\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \boldsymbol{P}^k\tilde{\boldsymbol{B}}^k.$$

- Random subspace P does not capture any problem structures
- Recall in coordinate gradient descent, there exists various greedy coordinate sampling strategies such as the *Gauss-Southwell* selection rules
- How to find better subspace P to enhance the performance of subspace methods?
- What structure shall the subspace P capture?

References

- Y. He, P. Li, Y. Hu, C. Chen, and **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

- Y. Chen, Y. Zhang, L. Cao, **K. Yuan**, Z. Wen*, Enhancing Zeroth-Order Fine-tuning for Language Models with Low-Rank Structures,* ICLR 2025

Thank you!