CrossLore: Boosting Low-Rank Adaptation for LLMs with Cross-Head Projection

Anonymous ACL submission

Abstract

Recent low-rank training methods, such as GaLore, have significantly reduced the memory required to optimize large language models (LLMs). However, these methods often suffer from time-consuming low-rank projection estimations. In particular, the singular value decomposition (SVD) in GaLore can consume more than 80% of the total training time. To address this issue, we propose CrossLore, which uses cross-head low-rank projection to reduce the substantial time consumption in estimating low-rank projections for multi-head attention. In addition, we employ randomized subspace iteration to achieve fast SVD. To further enhance performance, we propose sparsely coded residuals to reduce the errors caused by low-rank approximation on the first- and second-order moments of the optimizers and weight updates. We evaluate CrossLore on arithmetic reasoning and natural language generation datasets. Our experiments demonstrate that CrossLore delivers superior performance while achieving approximately $4\times$ fine-tuning speed compared to vanilla GaLore.

1 Introduction

As the sizes of language models grow rapidly, training models from scratch for different tasks becomes impractical due to the significant time and computational resources required. To address this challenge, current research and applications typically rely on pre-training large language models (LLMs) and subsequently fine-tuning them for specific downstream tasks. This paradigm has demonstrated high efficiency across various tasks, including NLP, question-answering, and reasoning (Roziere et al., 2023; Li et al., 2023; Ouyang et al., 2022; Brown, 2020).

However, full fine-tuning of entire LLMs requires enormous memory, making it prohibitively expensive for individuals and start-ups. Parameter-efficient fine-tuning (PEFT) methods only fine-tune a small number of the weights, significantly reducing the memory requirements (Lialin et al., 2023; Ding et al., 2023). Among them, the methods based on low-rank reparameterization, such as LoRA (Hu et al., 2022), have attracted much attention due to their impressive efficiency. Recently, a low-rank adaptation method named GaLore (Zhao et al., 2024) demonstrated the ability to optimize an LLM with 7 billion parameters on a consumer-level GPU with 24 GB of memory. Low-rank adaptation methods achieve dramatic memory reduction by updating the weights of LLMs in low-rank subspace, thus reducing the number of fine-tuned weights. We categorize existing low-rank adaptation methods into three groups based on how they construct low-rank projections *i.e.*, parameterized projection, random projection, and analytic projection.

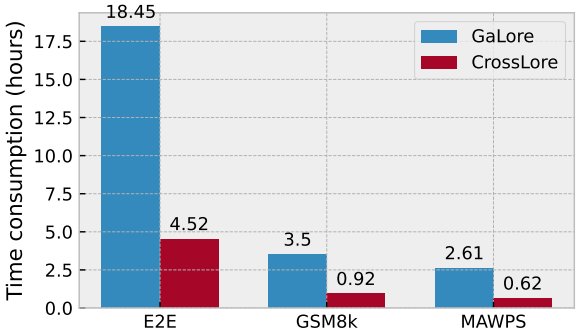


Figure 1: We compare the time consumption for fine-tuning LLaMA2-7B on different datasets with GaLore and CrossLore.

i) Parameterized projection. The representative work is LoRA (Hu et al., 2022), which approximates the parameter updates as the product of two trainable low-rank matrices. PiSSA (Meng et al., 2024) extends this by using SVD to decompose the weight matrix and optimizing the principal component with two trainable low-rank matrices, similar to LoRA. LoRA has been further developed into efficient variants like QLoRA (Dettmers et al., 2023),

which enhances computational efficiency and reduces memory through quantization. AdaLoRA enhances LoRA by adaptively adjusting the rank of low-rank updates (Zhang et al., 2023).

ii) Random projection. Methods in this category employ random projections to further reduce the memory consumption on parameterized projections. Flora (Hao et al., 2024) develops LoRA by substituting one of the two low-rank matrices with a randomly generated matrix, reducing the memory consumption with comparable performance. VeRA (Kopiczko et al., 2024) employs a shared pair of random projections across all layers, fine-tuning the model by training layer-specific scaling vectors.

iii) Analytic projection. Both parameterized and random projections may lead to low-quality approximations of gradient or weight updates, as they lack analytic guarantees. In contrast, GaLore (Zhao et al., 2024) introduces analytic projections derived from SVD to ensure that the key components of the gradients are preserved after low-rank projections, thereby offering a more accurate and reliable approximation. Tensor-GaLore (George et al., 2025) extends GaLore from matrix-based to tensor-based low-rank optimization. Meanwhile, WeLore (JAISWAL et al., 2025) extends GaLore by adaptively determining the number of retained singular values based on their heavy-tail distribution.

Existing methods for estimating low-rank projections involve a trade-off between approximation error and resource consumption (*e.g.*, on memory and time). Parameterized and random projections lack the accuracy of analytic decomposition, leading to uncertain quality, while analytic methods like SVD, though more precise, are computationally expensive. For instance, in GaLore, SVD accounts for over 80% of the total time consumed during the fine-tuning process. Therefore, a faster and more accurate estimation of low-rank projections is essential to further boost the low-rank adaptation methods.

In this paper, we propose a low-rank adaptation method, CrossLore, which employs cross-head low-rank projection to realize fast and high-quality estimation. Algorithm 1 presents the pseudo-code of integrating CrossLore into AdamW, with the highlighted improvement over GaLore (Zhao et al., 2024). CrossLore utilizes a cross-head low-rank projection inspired by the architecture of multi-head attention, where the projection matrices for the gradient of query or key transforms are shared across multiple attention heads. Such sharing reduces the computational complexity of low-rank projection matrices in h-head attention from $O(h^3)$ to $O(h)$. Additionally, randomized subspace iteration for SVD is utilized to further reduce computational complexity. Besides, CrossLore incorporates sparsely coded residuals, enabling a sparse representation of low-rank approximation errors in weight updates, which helps to mitigate estimation inaccuracies caused by the cross-head low-rank projection. We evaluate the proposed CrossLore on natural language processing and arithmetic reasoning tasks, comparing it against state-of-the-art low-rank adaptation methods.

The main contributions of this work are as follows:

- We introduce cross-head low-rank projection, which reduces computational complexity by sharing projection matrices across multiple query or key projections. Besides, we employ randomized subspace iteration to accelerate the estimation of the projections.
- We mitigate the impact of low-rank approximation errors on weight updates by utilizing sparsely coded residuals for the optimizer's moments, thereby enhancing the quality of the weight updates.
- Experimental results demonstrate that the proposed method surpasses state-of-the-art approaches, including LoRA and GaLore, in fine-tuning LLMs for tasks such as arithmetic reasoning and natural language generation.

2 Related Work

Parameter Efficient Fine-Tuning. A variety of parameter-efficient fine-tuning methods have emerged in recent years, enabling an increasing number of institutions and researchers to fine-tune LLMs to meet their specific requirements. Adapters (Rebuffi et al., 2017; Houlsby et al., 2019; Lin et al., 2020; Karimi Mahabadi et al., 2021b,a) enable parameter-efficient fine-tuning by inserting trainable layers into LLMs while keeping other layers frozen. However, this approach also introduces additional inference latency. BitFit (Zaken et al., 2021) selectively tunes only the biases within the network, significantly reducing the number of parameters involved in fine-tuning. Prompt tuning achieves parameter-efficient fine-tuning by

Algorithm 1 CrossLore (PyTorch-like pseudocode)

```
for weight in model.parameters():
    grad = weight.grad
    # original space -> compact space
    # cross-head low-rank projection
    lor_grad, lor_proj = project(grad)                          ▷ Section 4.1
    # update by AdamW or other optimizers
    lor_update, lor_moments = update(lor_grad)
    # sparsely coded residual
    res_update = estimate(grad, lor_proj, lor_moments)          ▷ Section 4.2
    # compact space -> original space
    update = project_back(lor_update) + res_update
    weight.data += update
```

Note: The green background highlights the improvements over GaLore (Zhao et al., 2024).

optimizing a set of new input tokens or prompts for each task (Li and Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2021; Liu et al., 2023). Hu et al. (2022) introduced LoRA, proposing that weight updates are low-rank and can be expressed as the product of two low-rank matrices. Furthermore, the trainable parameters can be merged with the original weights, eliminating additional inference latency. Recent studies combined parameter-efficient fine-tuning methods with quantization to enhance memory efficiency during the fine-tuning of LLMs (Kwon et al., 2022; Dettmers et al., 2023; Chai et al., 2023; Xu et al., 2023). And DoRA (Liu et al., 2024), or Weight-Decomposed Low-Rank Adaptation, is a parameter-efficient fine-tuning method designed to enhance learning capacity and stability by decomposing pre-trained weights into magnitude and direction components, leveraging LoRA for directional updates, and achieving superior performance across tasks without additional inference costs.

Parameter Sharing. Adam-mini partitions the model parameters into blocks based on the structure of the Hessian matrix, assigning a unified second-order moment to all parameters within each block (Zhang et al., 2024). This approach significantly reduces the memory footprint of the second-order moment, thereby decreasing the optimizer's memory usage. From a temporal perspective, GaLore shares the same projection across a fixed number of steps, reducing computational overhead. The above discussion demonstrates that many parameters can be shared during fine-tuning, reducing memory usage or computational complexity.

Low-Rank plus Sparse Matrix. Robust Principal Component Analysis (RPCA) decomposes a data matrix into the sum of the product of low-rank matrices and a sparse matrix and has been extensively studied in both theory and applications (Lin et al., 2010; Zhou and Tao, 2011; Liu et al., 2013; Aravkin et al., 2014; Hintermüller and Wu, 2015; Yi et al., 2016; Zhang and Yang, 2018). The recent Robust Adaptation (RoSA) method extends Low-Rank Adaptation (LoRA) by further decomposing weight updates into the product of two low-rank matrices, with an additional sparse matrix (Nikdan et al., 2024). Using an optimizer to update both the low-rank and sparse matrices, RoSA achieves superior performance compared to LoRA.

3 Preliminaries

GaLore. Conventional PEFT methods, such as LoRA (Hu et al., 2022), reduce the number of parameters for fine-tuning LLMs. However, the fixed low-rank nature of these methods limits the effectiveness of weight updates, resulting in performance inferior to full fine-tuning. GaLore (Zhao et al., 2024) addresses this limitation by leveraging the low-rank characteristics of gradients and projecting them onto low-rank subspace, significantly reducing the memory requirements for fine-tuning LLMs, while still maintaining the capability for full-parameter tuning. This approach enables pre-training an LLM with 7 billion parameters on a consumer-grade GPU, *i.e.*, NVIDIA RTX 4090 with 24 GB memory. The low-rank projections in GaLore are calculated via SVD and updated at fixed intervals. Thus, the search space for parameters can dynamically change within the full-rank space.

4 Methods

To reduce the time consumption of GaLore while improving performance, we propose CrossLore, which introduces two key components, *i.e.*, cross-head low-rank projection and sparsely coded residual. The cross-head low-rank projection enables efficient estimation of projection matrices with reduced computational complexity. Meanwhile, the sparsely coded residual corrects the weight update errors caused by low-rank projection, ensuring more accurate fine-tuning.

4.1 Cross-Head Low-Rank Projection

GaLore demonstrates remarkable performance on PEFT, enabling the training of a 7B LLM on a GPU with just 24GB of memory. However, the SVD employed in GaLore (Zhao et al., 2024) is inherently time-consuming. Figure 2a presents the time consumption for SVD and other operations during the fine-tuning of a LLaMA2-7B model using GaLore. The figure indicates that SVD accounts for more than 80% of the time consumption of the whole fine-tuning process. Moreover, SVD for the multi-head attention (MHA) layers alone takes about half of the fine-tuning process.

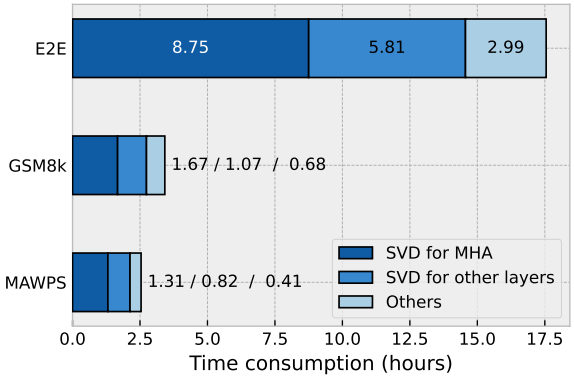
4.1.1 Cross-Head Similarity for Simplified SVD

Note that the three linear transformations across different heads share the same input and can be processed in parallel. Thus, in practical implementations, such as the PyTorch implementation[1], the linear transforms of different heads are concatenated into a single matrix. Here, we denote the *concatenated transforms* as

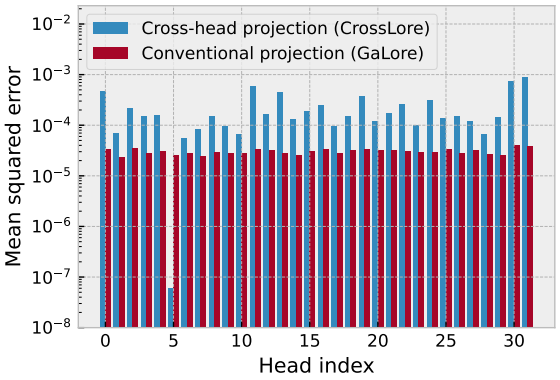
$$\begin{aligned} W^Q &\triangleq \left[W_1^Q, W_2^Q, \ldots, W_h^Q\right], \\ W^K &\triangleq \left[W_1^K, W_2^K, \ldots, W_h^K\right], \\ W^V &\triangleq \left[W_1^V, W_2^V, \ldots, W_h^V\right]. \end{aligned} \tag{1}$$

Existing low-rank based methods, such as LoRA (Hu et al., 2022) and GaLore (Zhao et al., 2024), conduct low-rank approximation on the updates of the concatenated multi-head transforms rather than on the updates of each individual head. Specifically, GaLore applies SVD to the gradient of a concatenated transform to get the low-rank projection. For instance, the r-rank ($r < hd_k$) approximation of the gradient $\nabla W^Q \in \mathbb{R}^{d_{\text{model}} \times hd_k}$ is

[1] https://pytorch.org/docs/stable/generated/torch.nn.MultiheadAttention.html



(a) Time consumption of GaLore



(b) Approximation error of low-rank projections

Figure 2: Motivations for cross-head low-rank projection. (a) illustrates the time consumption of SVD and other operations when fine-tuning an LLaMA2-7B model on different datasets with GaLore. MHA is short for multi-head attention. (b) presents the approximation errors of low-rank projection with cross-head projection (*i.e.*, CrossLore) and conventional projection (*i.e.*, GaLore).

is

$$\nabla W^Q = U\boldsymbol{\Sigma}V \approx U_{:,:r}\boldsymbol{\Sigma}_{:r,:r}V_{:r,:} = PP^\top \nabla W^Q, \tag{2}$$

where $P \triangleq U_{:,:r}$ is the low-rank projection matrix that contains the first r columns of U. The low-rank projections for ∇W^K and ∇W^V are calculated similarly.

Cordonnier et al. (2020) observed that the query or key transforms of different attention heads within the same layer are similar. The authors show that the concatenated projection is low-rank even though the projections of each head are of high ranks. Therefore, we hypothesize that the gradient ∇W_i^Q (or ∇W_i^K) of different heads within the same layer are similar. Thus, we can obtain a low-rank projection of the gradient of the concatenated transform ∇W^Q (or ∇W^K) via SVD on a

randomly selected W_i^Q (or ∇W_i^K), *i.e.*,

$$\begin{aligned}\nabla W_i^Q &= U_i \mathbf{\Sigma}_i V_i \approx (U_i)_{:,:r}(\mathbf{\Sigma}_i)_{:,:r}(V_i)_{:,:r} \\ &= P_i P_i^\top \nabla W_i^Q, \end{aligned} \tag{3}$$

where $P_i \triangleq (U_i)_{:,:r}$. Thus, the low-rank approximation of ∇W^Q in the proposed CrossLore is achieved by with

$$\nabla W^Q \approx P_i P_i^\top \nabla W^Q. \tag{4}$$

Figure 2b illustrates the approximation error of the gradients ∇W_i^Q with a randomly selected multi-head attention head using the low-rank projections using Equation 2 (vanilla GaLore) and Equation 4 (cross-head low-rank projection). The errors in Equation 4 are larger than those in Equation 2, as shown in Figure 2b. However, both errors remain relatively low. Moreover, existing research indicates that a certain degree of noise does not necessarily compromise the final performance of the model; in some cases, it may even enhance its robustness (Neelakantan et al., 2015; Li et al., 2020; Wu et al., 2020). The results presented in Figure 3 and Figure 4 further support this perspective. We further introduce sparsely coded residual to reduce for this discrepancy, as discussed in Subsection 4.2. Since the computational complexity of SVD is $O(mn \times \min(m, n))$ for a matrix of $m \times n$ elements, the computational complexity for the SVD operations of $\nabla W^Q \in \mathbb{R}^{d_{\text{model}} \times h d_k}$ and $\nabla W^K \in \mathbb{R}^{d_{\text{model}} \times h d_k}$ can be reduced from $O(h^3 d_k^3)$ to $O(h d_k^3)$. As a reference, h is set to 32 in LLaMA2-7B.

4.1.2 Fast SVD with Randomized Subspace Iteration

To further reduce the time consumption on SVD, we adopt the *randomized subspace iteration* algorithm proposed by Halko et al. (2011), which is a fast implementation of SVD. To obtain the $m \times r$ low-rank projection matrix from an $m \times n$ matrix, the randomized subspace iteration can reduce the computational complexity from $O(mn \times \min(m, n))$ to $O(mn \times \log(r))$. Specifically, the computational complexity for the SVD operations of ∇W^Q and ∇W^K can be reduced from $O(h d_k^3)$ to $O(h d_k^2 \times \log(r))$.

Furthermore, randomized subspace iteration also reduces memory consumption during fine-tuning. SVD of an $m \times n$ matrix produces three matrices with the shapes of $m \times m$, $\min(m, n)$, and $n \times n$. However, we only use one matrix of $m \times m$ or $n \times n$ to get the low-rank projection matrix. Randomized subspace iteration only produces a $r \times r$ matrix during SVD, which consumes significantly less memory. The experiments in Section 5 demonstrate the advantages of randomized subspace iteration regarding time efficiency and memory usage.

4.2 Sparsely Coded Residual

The low-rank projection of the gradients can significantly reduce the memory consumption during fine-tuning, along with the low-rank first- and second-order moments for optimizers. However, the low-rank projection may not always be the main component of practical implementations. For instance, the low-rank projections are updated every hundred steps during fine-tuning due to the high computational complexity of SVD, making it challenging to apply low-rank projections at every step. Moreover, the cross-head low-rank projection introduces increased approximation error due to cross-head sharing and randomized subspace iterations, leading to less accurate SVD results. Consequently, the low-rank first- and second-order moments for the optimizers are also imprecise. To address this, we estimate the residuals using a sparse representation, which improves the quality of the first- and second-order moments.

4.2.1 Low-Rank Approximation Residual of Moments

Let $G_t \in \mathbb{R}^{m \times n}$ be the gradient of t-th step, and $P_t \in \mathbb{R}^{m \times r}$ be the low-rank projection of t-th step, the residual of low-rank approximation of G_t is

$$\Delta G_t = G_t - P_t P_t^\top G_t. \tag{5}$$

The first- and second-order moments (denoted as M_t and V_t, with $M_t, V_t \in \mathbb{R}^{m \times n}$) for common optimizers, such as Adam, AdaGrad, and AdamW, are estimated as

$$\begin{aligned} M_t &= \beta_1 M_{t-1} + (1 - \beta_1) G_t, \\ V_t &= \beta_2 V_{t-1} + (1 - \beta_2) G_t \odot G_t, \end{aligned} \tag{6}$$

where β_1 and β_2 are decay rates of the moments, and $\odot$ means element-wise multiplication. The low-rank first- and second-order moments (denoted as M_t' and V_t', with $M_t', V_t' \in \mathbb{R}^{r \times n}$) used in GaLore are realized by

$$\begin{aligned} M_t' &= \beta_1 M_{t-1}' + (1 - \beta_1) P_t^\top G_t, \\ V_t' &= \beta_2 V_{t-1}' + (1 - \beta_2)(P_t^\top G_t) \odot (P_t^\top G_t). \end{aligned} \tag{7}$$

The low-rank approximation residuals of the moments are

$$\Delta M_t = M_t - P_t M_t', \ \Delta V_t = V_t - P_t V_t'. \quad (8)$$

Equation 8 can be extended into the following form considering Equation 5, 6, and 7,

$$\begin{aligned} \Delta M_t &\approx \beta_1 \Delta M_{t-1} + (1-\beta_1)\Delta G_t, \\ \Delta V_t &\approx \beta_2 \Delta V_{t-1} + 2(1-\beta_2)(P_t P_t^\top G_t) \odot \Delta G_t, \end{aligned} \quad (9)$$

The detailed inference is provided in Appendix A. Equation 9 depicts the evolution of the low-rank approximation residual of the moments during fine-tuning. Thus, we employ two additional variables in fine-tuning as the estimate of the low-rank approximation residual, and the two variables update following Equation 9.

Furthermore, the approximation residual of the moments leads to bias in parameter updates. The bias formulation depends on the specific form of the optimizer, and we use AdamW as an example. Let ΔW_t and $\Delta W_t'$ be the full-rank parameter update and the low-rank reconstruction of W at step t, then the following equations hold.

$$\Delta W_t = \frac{M_t / \left(1-\beta_1^t\right)}{\sqrt{V_t / \left(1-\beta_2^t\right)} + \epsilon}, \quad (10)$$

$$\Delta W_t' = \frac{P_t M_t' / \left(1-\beta_1^t\right)}{\sqrt{P_t V_t' / \left(1-\beta_2^t\right)} + \epsilon}. \quad (11)$$

Then the low-rank approximation error of the update, denoted as δ_t, is

$$\begin{aligned} \delta_t &= \Delta W_t - \Delta W_t' \\ &\approx \frac{\Delta M_t / \left(1-\beta_1^t\right)}{\sqrt{(\Delta V_t + P_t V_t') / \left(1-\beta_2^t\right)} + \epsilon}. \end{aligned} \quad (12)$$

Equation 12 demonstrates the evolution of low-rank approximation error of the update during fine-tuning, which is also used in the proposed sparsely coded residual.

4.2.2 Sparse Indexing Matrix for the Residuals

We can leverage the residuals in Equation 9 and Equation 12 to improve the quality of the updates during the optimization process. However, the residuals in Equation 9 and 12 are full-rank matrices, *i.e.*, $\Delta M_t, \Delta V_t, \delta_t \in \mathbb{R}^{m \times n}$, consuming enormous memory during fine-tuning. To incorporate the residuals in memory-efficient PEFT methods, we employ sparse representations with a sparse indexing matrix preserving the most significant elements of the residuals.

Algorithm 2 Sparsely Coded Residual of AdamW

given time step t, gradient $G_t \in \mathbb{R}^{m \times n}$, sparse indexing matrix $L \in \mathbb{R}^{m \times n}$, low-rank projection $P_t \in \mathbb{R}^{m \times r}$, second-order low-rank moment $V_t' \in \mathbb{R}^{r \times n}$, first- and second-order moment residuals $\Delta M_{t-1}, \Delta V_{t-1} \in \mathbb{R}^{m \times n}$, constant ϵ
$\hat{G}_t \leftarrow P_t P_t^\top G_t$
$\Delta G_t \leftarrow (G_t - \hat{G}_t) \odot L$
$\Delta M_t \leftarrow \beta_1 \Delta M_{t-1} + (1-\beta_1)\Delta G_t$
$\Delta V_t \leftarrow \beta_2 \Delta V_{t-1} + 2(1-\beta_2)\hat{G}_t \odot \Delta G_t$
$\Delta \hat{M}_t \leftarrow \Delta M_t / (1-\beta_1^t)$
$\Delta \hat{V}_t \leftarrow \Delta V_t / (1-\beta_2^t)$
$\delta_t \leftarrow \Delta \hat{M}_t / \left(\sqrt{P_t V_t' / (1-\beta_2^t) + \Delta \hat{V}_t} + \epsilon \right)$
return compact residual δ_t

We introduce a warm-up stage to determine the sparse indexing matrix essential for efficient fine-tuning. This warm-up stage occurs during the first k steps of the fine-tuning process. At the end of the warm-up stage, the positions for the top-1% absolute values in the reconstructed first-order moment (*i.e.*, $P_t M_t'$) are recorded and used to create the sparse indexing matrix. This allows the approximation residuals for both moments and updates to be stored in sparse matrices, significantly reducing memory requirements. During the warm-up stage, the residuals are set to 0.

Algorithm 2 presents the whole process for obtaining the compact residuals at the t-th ($t > k$) step of AdamW. Please note that the proposed method can also be incorporated into other optimizers with moments, such as Adam and AdaGrad, to improve the quality of updates with low-rank approximation.

5 Experiments

In this section, we present a series of experiments to evaluate the effectiveness of CrossLore. We compare our proposed method against a range of baselines in fine-tuning the LLaMA2-7B and LLaMA2-13B models (Touvron et al., 2023), specifically focusing on tasks related to arithmetic reasoning and natural language generation, to assess its overall performance.

Implementation Details. We fine-tune all lay-

Datasets		GSM8k			MAWPS		
Methods	Rank	Time↓	Mem.↓	Acc. (%)↑	Time↓	Mem.↓	Acc. (%)↑
LoRA	16	0.53	15.40	25.55	**0.38**	14.11	47.90
DoRA	16	1.12	**14.76**	22.84	0.68	**13.88**	47.48
GaLore	16	3.48	15.33	26.31	2.58	15.06	62.61
CrossLore	16	0.88	15.10	**27.47**	0.62	14.84	**63.87**
LoRA	128	0.53	16.99	30.78	**0.45**	15.64	65.97
DoRA	128	1.18	16.17	29.36	0.72	16.17	66.81
GaLore	128	3.50	16.06	33.66	2.61	15.79	64.29
CrossLore	128	0.92	**15.73**	**34.65**	0.62	**15.44**	**67.64**

Table 1: Comparison of fine-tuning LLaMA2-7B with different PEFT methods on the GSM8k and MAWPS datasets. The metrics for fine-tuning time (Time) and memory consumption (Mem.) are 'hours' and 'GB'.

ers of the LLaMA2-7B and LLaMA2-13B models, adding sparsely coded residuals only in the query and key projections, and load the parameters in `bfloat16` format. For the arithmetic reasoning task, we evaluate the accuracy on the test set, while for the natural language generation task, we measure both the similarity and quality of the generated text compared to the reference text. All these experiments on LLaMA2-7B are carried out on an NVIDIA RTX 4090 GPU with 24GB of memory, using the Llama-Factory framework (Zheng et al., 2024) for implementation. Additionally, experiments on LLaMA2-13B are conducted on an NVIDIA A800 GPU.

Baselines. We apply CrossLore with the following baseline methods:

i) LoRA (Hu et al., 2022) enables efficient model adaptation by freezing the backbone network and optimizing only low-rank adapters.

ii) GaLore (Zhao et al., 2024) is a memory-efficient full-parameter fine-tuning method that significantly reduces memory usage by projecting gradients onto a low-rank subspace.

iii) DoRA (Liu et al., 2024) builds on LoRA by decomposing weights into magnitude and direction, enhancing learning efficiency and stability.

5.1 Arithmetic Reasoning

Setups. For the arithmetic reasoning task, we utilize the GSM8k and MAWPS datasets to fine-tune and evaluate the models (Cobbe et al., 2021; Koncel-Kedziorski et al., 2016). We set the rank r to 16, 32, 128 and 256 for LoRA, DoRA, GaLore, and CrossLore. Detailed hyperparameter settings are provided in the Appendix B.

Main Results. Table 1 compares the performance of CrossLore with other PEFT methods on the LLaMA2 models. On the GSM8k dataset, CrossLore outperforms other PEFT methods in accuracy at ranks. For instance, at rank 128, CrossLore achieves an accuracy of 34.65% on the GSM8k dataset, when fine-tuning LLaMA2-7B, surpassing the second-best result of 33.66%. Similar trends are observed on the MAWPS dataset, further demonstrating the superior performance of CrossLore. The experimental results for LLaMA2-7B with ranks 32 and 256, as well as those for LLaMA2-13B with ranks 16, 32, 128, and 256, are provided in the Appendix C.

5.2 Natural Language Generation

Setups. For the natural language generation task, we fine-tune and evaluate the model using the E2E dataset (Novikova et al., 2017). We set the rank r to 16, 32, 128 and 256 for thorough comparison. We set the number of training epochs to 1 and set the learning rate to 1×10^{-6}. We compare the performance of LoRA, DoRA, GaLore, and the proposed CrossLore with a range of metrics, including peak memory consumption, BLEU, NIST, MET, ROUGE-1/2/L, and CIDEr. For detailed experiment settings, please refer to Appendix B.

Main Results. Table 2 presents the experimental results on the E2E dataset, covering various metrics, with 'Mem.' representing peak memory consumption. Notably, CrossLore exhibits a competitive or superior performance in most metrics compared to LoRA, DoRA and GaLore. For instance, with rank 128, CrossLore achieves better results in terms of NIST, ROUGE, and CIDEr scores while maintaining the same memory usage as LoRA, when fine-tuning LLaMA2-7B. The experimental results for LLaMA2-7B with ranks 32 and 256, as well as those for LLaMA2-13B with ranks 16, 32, 128, and 256, are provided in the Appendix C. To further validate the capabilities of CrossLore, we fine-tuned

Methods	Rank	Time↓	Mem.↓	BLEU↑	NIST↑	MET↑	ROUGE-1/2/L↑	CIDEr↑
LoRA	16	2.82	13.92	60.29	4.00	34.91	66.24 / 34.70 / 43.15	1.20
DoRA	16	4.97	**13.88**	60.39	4.05	34.93	66.26 / 34.74 / 43.30	1.21
GaLore	16	18.53	15.07	64.84	**4.96**	36.60	70.79 / 40.62 / 48.90	**1.78**
CrossLore	16	4.38	14.83	**65.16**	**4.96**	**36.94**	**70.98 / 40.85 / 49.05**	**1.78**
LoRA	128	3.02	**15.43**	64.60	4.86	36.74	70.67 / 40.20 / 48.54	1.70
DoRA	128	5.27	**15.43**	**64.77**	4.86	**36.77**	70.72 / 40.20 / 48.62	1.72
GaLore	128	18.45	15.79	64.22	5.14	36.50	70.99 / 41.19 / 49.37	1.80
CrossLore	128	4.52	**15.43**	64.22	**5.16**	36.66	**71.07 / 41.54 / 49.71**	**1.82**

Table 2: Comparison of fine-tuning LLaMA2-7B with different PEFT methods on the E2E dataset. The metrics for fine-tuning time (Time) and memory consumption (Mem.) are 'hours' and 'GB'.

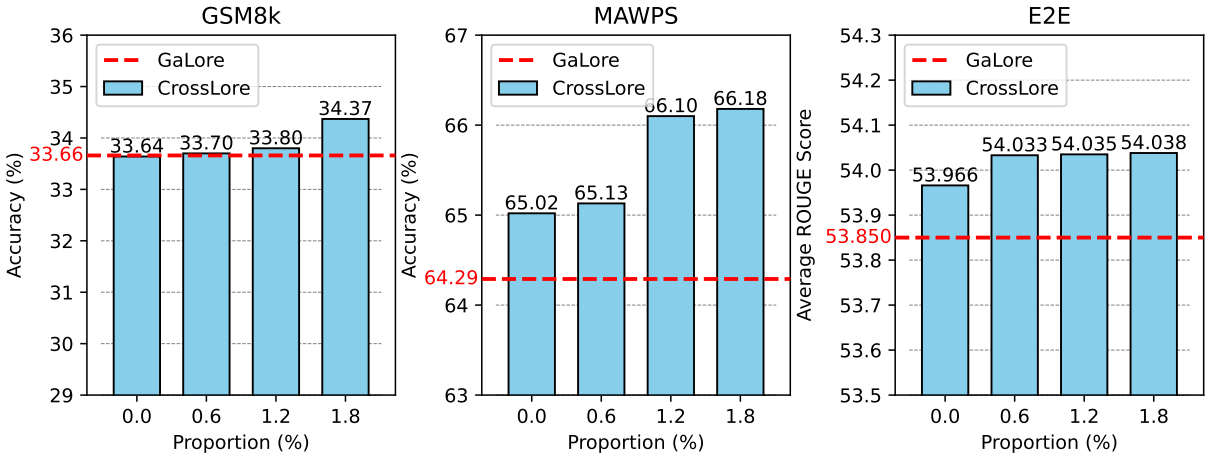


Figure 3: Ablation study on the sparsely coded residual, when fine-tuning LLaMA2-7B
. The results are obtained by adjusting the proportion of non-zero elements in the sparse indexing matrix.

LLaMA2-7B and LLaMA2-13B on the CNN/DM and XSum datasets, comparing it against several other PEFT methods. Detailed results are provided in Appendix C.

Additionally, Figure 1 compares the time consumption between GaLore and CrossLore, when fine-tuning LLaMA2-7B. One of the key advantages of CrossLore is the negligible time to compute the projection matrices. As a result, the overall fine-tuning time is reduced by more than 70%, significantly accelerating the training process without compromising performance.

5.3 Ablation Study

To evaluate the effectiveness of the proposed sparsely coded residual, we conduct experiments on CrossLore with increasing number of non-zero elements in the sparse indexing matrix. All hyperparameters except for the sparsity of the residuals are kept unchanged for reasonable ablation. For robustness of the results, experiments were conducted with four random seeds set for each experiment. The experimental results for LLaMA3-8B are provided in Appendix D.

Figure 3 presents the results of the comparison, suggesting that the residuals with more non-zero elements lead to better performance in most cases which indicates that the sparsely coded residuals indeed mitigate projection-induced errors and enhance the model's performance.

6 Conclusions

In this paper, we propose CrossLore, a low-rank adaptation method with fast yet precise estimation of the low-rank projections. The estimation is achieved by cross-head low-rank projection and randomized subspace iteration. We further employ a sparsely coded residual to further reduce the error of low-rank approximation, which works on the moments of the optimizer. Experiments on arithmetic reasoning and natural language generation indicate that CrossLore outperforms existing low-rank adaptation methods, offering superior performance with reduced computational complexity.

Limitations

There are two limitations in this work. Firstly, although the experiments validate that sharing low-rank projection matrices across heads in multi-head attention can be efficient, additional theoretical analysis is needed to measure the preciseness of such sharing. Secondly, we employ sparse matrices to store the approximation errors of first- and second-order moments and weight updates. However, all these sparse matrices share the same indexing matrices. Although experiments indicate that the current design is effective, the shared indexing matrices estimated from the first-order moment can be further improved.

References

Aleksandr Aravkin, Stephen Becker, Volkan Cevher, and Peder Olsen. 2014. A variational approach to stable principal component pursuit. *arXiv preprint arXiv:1406.1089*.

Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Yuji Chai, John Gkountouras, Glenn G Ko, David Brooks, and Gu-Yeon Wei. 2023. Int2. 1: Towards fine-tunable quantized large language models with error correction through low-rank adaptation. *arXiv preprint arXiv:2306.08162*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. 2020. Multi-head attention: Collaborate instead of concatenate. *arXiv preprint arXiv:2006.16362*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235.

Robert Joseph George, David Pitt, Jiawei Zhao, Jean Kossaifi, Cheng Luo, Yuandong Tian, and Anima Anandkumar. 2025. Tensor-galore: Memory-efficient training via gradient tensor decomposition.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. Warp: Word-level adversarial reprogramming. *arXiv preprint arXiv:2101.00121*.

Yongchang Hao, Yanshuai Cao, and Lili Mou. 2024. Flora: Low-rank adapters are secretly gradient compressors. In *Forty-first International Conference on Machine Learning*.

Michael Hintermüller and Tao Wu. 2015. Robust principal component pursuit via inexact alternating minimization on matrix manifolds. *Journal of Mathematical Imaging and Vision*, 51(3):361–377.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

AJAY KUMAR JAISWAL, Lu Yin, Zhenyu Zhang, Shiwei Liu, Jiawei Zhao, Yuandong Tian, and Zhangyang Wang. 2025. From galore to welore: How low-rank weights non-uniformly emerge from low-rank gradients.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021a. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021b. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Annual Meeting of the Association for Computational Linguistics*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, pages 1152–1157.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. 2024. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*.

Se Jung Kwon, Jeonghoon Kim, Jeongin Bae, Kang Min Yoo, Jin-Hwa Kim, Baeseong Park, Byeongwook Kim, Jung-Woo Ha, Nako Sung, and Dongsoo Lee. 2022. AlphaTuning: Quantization-aware parameter-efficient adaptation of large-scale pre-trained language models. In *Findings of the Association for*

Computational Linguistics: EMNLP 2022, pages 3288–3305, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Xingjian Li, Haoyi Xiong, Haozhe An, Cheng-Zhong Xu, and Dejing Dou. 2020. Rifle: Backpropagation in depth for deep transfer learning through re-initializing the fully-connected layer. In *International Conference on Machine Learning*, pages 6010–6019. PMLR.

Yunxiang Li, Zihan Li, Kai Zhang, Ruilong Dan, Steve Jiang, and You Zhang. 2023. Chatdoctor: A medical chat model fine-tuned on a large language model meta-ai (llama) using medical domain knowledge. *Cureus*, 15(6).

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. 2023. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. 2020. Exploring versatile generative language model via parameter-efficient transfer learning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 441–459, Online. Association for Computational Linguistics.

Zhouchen Lin, Minming Chen, and Yi Ma. 2010. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *arXiv preprint arXiv:1009.5055*.

Guangcan Liu, Zhouchen Lin, Shuicheng Yan, Ju Sun, Yong Yu, and Yi Ma. 2013. Robust recovery of subspace structures by low-rank representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(1):171–184.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2023. Gpt understands, too. *AI Open*.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024. Pissa: Principal singular values and singular vectors adaptation of large language models. *arXiv preprint arXiv:2404.02948*.

Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. 2015. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*.

Mahdi Nikdan, Soroush Tabesh, and Dan Alistarh. 2024. Rosa: Accurate parameter-efficient fine-tuning via robust adaptation. *arXiv preprint arXiv:2401.04679*.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The E2E dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 201–206, Saarbrücken, Germany. Association for Computational Linguistics.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. *Advances in neural information processing systems*, 30.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Jingfeng Wu, Wenqing Hu, Haoyi Xiong, Jun Huan, Vladimir Braverman, and Zhanxing Zhu. 2020. On the noisy gradient descent that generalizes as sgd. In *International Conference on Machine Learning*, pages 10367–10376. PMLR.

Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. 2023. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*.

Xinyang Yi, Dohyung Park, Yudong Chen, and Constantine Caramanis. 2016. Fast algorithms for robust pca via gradient descent. *Advances in neural information processing systems*, 29.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Teng Zhang and Yi Yang. 2018. Robust pca by manifold optimization. *Journal of Machine Learning Research*, 19(80):1–39.

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. 2024. Adam-mini: Use fewer learning rates to gain more. *arXiv preprint arXiv:2406.16793*.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. Galore: Memory-efficient LLM training by gradient low-rank projection. In *5th Workshop on practical ML for limited/low resource settings*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372*.

Tianyi Zhou and Dacheng Tao. 2011. Godec: Randomized low-rank & sparse matrix decomposition in noisy case. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*.

A Proof of Sparsely Coded Residual

The origin of errors and the calculation of sparsely coded residual are analyzed, using AdamW as an example.

First-order moment (denoted as M_t, with $M_t \in \mathbb{R}^{m\times n}$) can be regarded as the sum of first-order moment projected back onto the original space and the error (denoted as ΔM_t, with $\Delta M_t \in \mathbb{R}^{m\times n}$). Similarly, the gradient (denoted as G_t, with $G_t \in \mathbb{R}^{m\times n}$) can be viewed as the sum of the gradient projected back onto the original space and the error (denoted as ΔG_t, with $\Delta G_t \in \mathbb{R}^{m\times n}$):

$$M_t = P_t P_t^\top M_t + \Delta M_t, \tag{13}$$

$$G_t = P_t P_t^\top G_t + \Delta G_t, \tag{14}$$

where P_t represents the projection matrix at step t. Therefore, the first-order moment updates are shown in Equation 15.

$$\begin{aligned} M_t =& \beta_1 M_{t-1} + (1-\beta_1) G_t \\ =& \beta_1 P_{t-1} P_{t-1}^\top M_{t-1} + (1-\beta_1) P_t P_t^\top G_t \\ &+ \beta_1 \Delta M_{t-1} + (1-\beta_1)\Delta G_t, \end{aligned} \tag{15}$$

The low-rank first-order moments (denoted as $M_t' = P_t^\top M_t$, with $M_t' \in \mathbb{R}^{r\times n}$) used in GaLore are realized by:

$$M_t' = \beta_1 M_{t-1}' + (1-\beta_1) P_t^\top G_t. \tag{16}$$

The low-rank approximation error of the first-order moment is

$$\begin{aligned} \Delta M_t =& M_t - P_t M_t' \\ =& \beta_1 (P_{t-1} - P_t) P_{t-1}^\top M_{t-1} + \beta_1 \Delta M_{t-1} \\ &+ (1-\beta_1)\Delta G_t. \end{aligned} \tag{17}$$

The projection matrix P_t is updated every few hundred steps, thus, outside of these updates, P_{t-1} remains equal to P_t. When P_t is updated, the weight updates ΔW_t are so small that $P_{t-1} \approx P_t$, similar to the discussion in Section 4.1.1. Therefore, the low-rank approximation error of the first-order moment is

$$\Delta M_t \approx \beta_1 \Delta M_{t-1} + (1-\beta_1)\Delta G_t. \tag{18}$$

Similarly, the update process for the second-order moment is as follows:

$$V_t = P_t P_t^\top V_t + \Delta V_t, \tag{19}$$

$$\begin{aligned} V_t =& \beta_2 V_{t-1} + (1-\beta_2) G_t \odot G_t \\ =& \beta_2 P_{t-1} P_{t-1}^\top V_{t-1} \\ &+ (1-\beta_2)(P_t P_t^\top G_t) \odot (P_t P_t^\top G_t) + \beta_2 \Delta V_{t-1} \\ &+ (1-\beta_2)\left[\Delta G_t \odot \Delta G_t + 2(P_t P_t^\top G_t) \odot \Delta G_t\right], \end{aligned} \tag{20}$$

$$V_t' = \beta_2 V_{t-1}' + (1-\beta_2)(P_t^\top G_t) \odot (P_t^\top G_t), \tag{21}$$

$$\begin{aligned} \Delta V_t =& V_t - P_t V_t' \\ \approx& \beta_2 \Delta V_{t-1} + 2(1-\beta_2)(P_t P_t^\top G_t) \odot \Delta G_t \\ &+ (1-\beta_2)\big[\Delta G_t \odot \Delta G_t \\ &+ (P_t P_t^\top G_t) \odot (P_t P_t^\top G_t) \\ &- P_t (P_t^\top G_t) \odot (P_t^\top G_t)\big] \\ \approx& \beta_2 \Delta V_{t-1} + 2(1-\beta_2)(P_t P_t^\top G_t) \odot \Delta G_t \\ &+ (1-\beta_2)\big[(P_t P_t^\top G_t) \odot (P_t P_t^\top G_t) \\ &- P_t (P_t^\top G_t) \odot (P_t^\top G_t)\big]. \end{aligned} \tag{22}$$

ΔV_t can be approximated as 0. The expression $(P_t P_t^\top G_t) \odot (P_t P_t^\top G_t) - P_t (P_t^\top G_t) \odot (P_t^\top G_t)$ is only a part of ΔV_t, and its value is quite small. Moreover, considering the significant computational burden associated with this part, we neglect its calculation:

$$\Delta V_t \approx \beta_2 \Delta V_{t-1} + 2(1-\beta_2)(P_t P_t^\top G_t) \odot \Delta G_t.$$

Thus, the parameter update can be expressed as

follows:

$$\begin{aligned}\Delta W_t =& \frac{M_t/(1-\beta_1^t)}{\sqrt{V_t/(1-\beta_2^t)}+\epsilon} \\ =& \frac{\Delta\hat{M}_t + \hat{M}_t}{\sqrt{\Delta\hat{V}_t + \hat{V}_t}+\epsilon} \\ =& \frac{\hat{M}_t}{\sqrt{\hat{V}_t}+\epsilon} + \left[\hat{M}_t\left(\sqrt{\hat{V}_t} - \sqrt{\Delta\hat{V}_t+\hat{V}_t}\right)\right. \\ & \left. + \Delta\hat{M}_t\left(\sqrt{\hat{V}_t}+\epsilon\right)\right] \\ & \Big/\left[\left(\sqrt{\Delta\hat{V}_t+\hat{V}_t}+\epsilon\right)\left(\sqrt{\hat{V}_t}+\epsilon\right)\right],\end{aligned} \tag{23}$$

where

$$\Delta\hat{M}_t = \Delta M_t/\left(1-\beta_1^t\right), \hat{M}_t = P_t M_t'/\left(1-\beta_1^t\right),$$
$$\Delta\hat{V}_t = \Delta V_t/\left(1-\beta_2^t\right), \hat{V}_t = P_t V_t'/\left(1-\beta_2^t\right),$$

$\Delta\hat{V}_t$ can be approximated as 0, so the Equation 23 can be approximated as:

$$\begin{aligned}\Delta W_t &\approx \frac{\hat{M}_t}{\sqrt{\hat{V}_t}+\epsilon} + \frac{\Delta\hat{M}_t}{\sqrt{\Delta\hat{V}_t+\hat{V}_t}+\epsilon} \\ &= \frac{\hat{M}_t}{\sqrt{\hat{V}_t}+\epsilon} + \delta_t,\end{aligned} \tag{24}$$

where δ_t is the residual neglected by GaLore:

$$\delta_t = \frac{\Delta\hat{M}_t}{\sqrt{\Delta\hat{V}_t+\hat{V}_t}+\epsilon}. \tag{25}$$

B Details of Experiment

We fine-tuned the LLaMA2-7B model to validate the effectiveness of CrossLore. All experiments were conducted on an NVIDIA RTX 4090 GPU using the Llama-Factory framework. Detailed hyperparameter settings are provided in Table 3.

We fine-tuned the LLaMA2-13B model to validate the effectiveness of CrossLore. All experiments were conducted on an NVIDIA A800 GPU using the Llama-Factory framework. Detailed hyperparameter settings are provided in Table 4.

C Additional Experimental Results

We have conducted additional experiments to investigate the effectiveness of CrossLore. We have included two foundation models (i.e., LLaMA2-7B and LLaMA2-13B), and state-of-the-art PEFT methods as the baseline, with detailed experimental results presented in Table 5, 6, 7, 8 and 9.

As shown in Table 5, 6, 7, 8 and 9, CrossLore consistently achieves comparable or superior results across various scenarios. Experimental results show that the proposed method is specifically effective in low-data fine-tuning scenarios, which outperforms LoRA and other fine-tuning methods significantly.

D Results of Ablation Study

To validate the effectiveness of the sparsely coded residual, we also fine-tuned LLaMA3-8B with residuals incorporated at varying proportions. The experimental results are presented as shown in Figure 4

The experimental results in Figure 3 and Figure 4 were obtained using 4 random seeds, with the standard deviations reported in Table 10.

Dataset	E2E				GSM8k				MAWPS			
Rank	16	32	128	256	16	32	128	256	16	32	128	256
Batch Size	1				1				1			
Epochs	1				1				3			
LearningRate	1E-06				1E-06				1E-06			
LR Schedul	cosine				cosine				cosine			
α	32	32	128	64	32	32	64	32	32	16	32	32
Optimizer	AdamW8bit				AdamW8bit				AdamW8bit			
Residual Ratio	1.2%				1.2%				1.2%			

Table 3: Hyperparameters on three benchmarks for the LLaMA2-7B model.

Dataset	E2E				GSM8k				MAWPS			
Rank	16	32	128	256	16	32	128	256	16	32	128	256
Batch Size	4				1				1			
Epochs	1				1				3			
Learning Rate	1E-06				1E-06				1E-06			
LR Schedule	cosine				cosine				cosine			
α	32	32	128	64	32				32	32	64	64
Optimizer	AdamW8bit				AdamW8bit				AdamW8bit			
Residual Ratio	1.2%				1.2%				1.2%			

Table 4: Hyperparameters on three benchmarks for the LLaMA2-13B model.

Datasets		**GSM8k**			**MAWPS**		
Methods	Rank	Time↓	Mem.↓	Acc. (%)↑	Time↓	Mem.↓	Acc. (%)↑
LoRA	32	0.53	15.65	23.30	0.40	**14.36**	45.80
DoRA	32	1.15	**15.01**	21.08	0.69	15.01	44.96
GaLore	32	3.48	15.42	26.46	2.59	15.15	58.40
CrossLore	32	0.88	15.23	**29.11**	0.62	14.91	**60.50**
LoRA	256	0.55	18.79	33.36	0.4	17.60	61.76
DoRA	256	1.24	18.12	33.59	0.76	18.12	62.18
GaLore	256	3.53	16.66	31.92	2.66	16.39	63.03
CrossLore	256	0.92	**15.74**	**33.81**	0.69	**16.12**	**65.55**

Table 5: Comparison of fine-tuning LLaMA2-7B with different PEFT methods on the GSM8k and MAWPS datasets at ranks 32 and 256. The metrics for fine-tuning time (Time) and memory consumption (Mem.) are 'hours' and 'GB'.

Datasets		GSM8k			MAWPS		
Methods	Rank	Time↓	Mem.↓	Acc. (%)↑	Time↓	Mem.↓	Acc. (%)↑
LoRA	16	0.73	27.59	30.46	0.50	**26.14**	54.20
DoRA	16	1.70	**26.67**	31.22	1.04	26.21	54.20
GaLore	16	7.18	28.01	35.18	5.39	27.82	58.40
CrossLore	16	1.22	27.83	**36.24**	0.79	27.49	**62.18**
LoRA	32	0.73	27.96	32.01	0.50	**26.50**	55.04
DoRA	32	1.68	**27.04**	32.12	1.03	26.57	53.78
GaLore	32	7.08	28.14	36.24	5.34	27.94	61.34
CrossLore	32	1.22	27.93	**37.98**	0.80	27.59	**63.87**
LoRA	128	0.77	30.11	33.99	0.53	28.77	54.62
DoRA	128	1.78	28.98	32.68	1.09	28.70	54.62
GaLore	128	7.20	29.28	40.03	5.39	28.99	65.13
CrossLore	128	1.37	**28.90**	**41.24**	0.91	**28.54**	**69.75**
LoRA	256	0.80	33.08	34.72	0.55	31.62	62.18
DoRA	256	1.83	31.88	33.59	1.14	31.72	62.18
GaLore	256	7.25	30.06	37.15	5.50	29.87	62.18
CrossLore	256	1.54	**29.79**	**42.20**	1.06	**29.45**	**66.39**

Table 6: Comparison of fine-tuning LLaMA2-13B with different PEFT methods on the GSM8k and MAWPS datasets. The metrics for fine-tuning time (Time) and memory consumption (Mem.) are 'hours' and 'GB'.

Methods	Rank	Time↓	Mem.↓	BLEU↑	NIST↑	MET↑	ROUGE-1/2/L↑	CIDEr↑
LoRA	32	2.94	14.17	62.47	4.58	35.07	69.16 / 38.00 / 46.24	1.46
DoRA	32	5.14	**14.13**	62.63	4.59	35.11	69.25 / 38.07 / 46.31	1.47
GaLore	32	18.57	15.15	64.95	4.96	36.92	70.99 / 40.77 / 49.15	1.77
CrossLore	32	4.41	14.92	**65.15**	**4.97**	**37.32**	**71.17 / 41.06 / 49.38**	**1.78**
LoRA	256	3.01	17.43	64.93	4.94	36.81	70.92 / 40.66 / 49.01	1.76
DoRA	256	5.61	17.67	64.94	4.95	36.81	70.94 / 40.72 / 49.04	1.77
GaLore	256	18.42	16.39	64.97	5.10	36.84	**71.19** / 41.34 / 49.26	1.82
CrossLore	256	4.86	**16.19**	**64.99**	**5.11**	**36.88**	71.15 / **41.40 / 49.45**	**1.83**

Table 7: Comparison of fine-tuning LLaMA2-7B with different PEFT methods on the E2E dataset at ranks 32 and 256. The metrics for fine-tuning time (Time) and memory consumption (Mem.) are 'hours' and 'GB'.

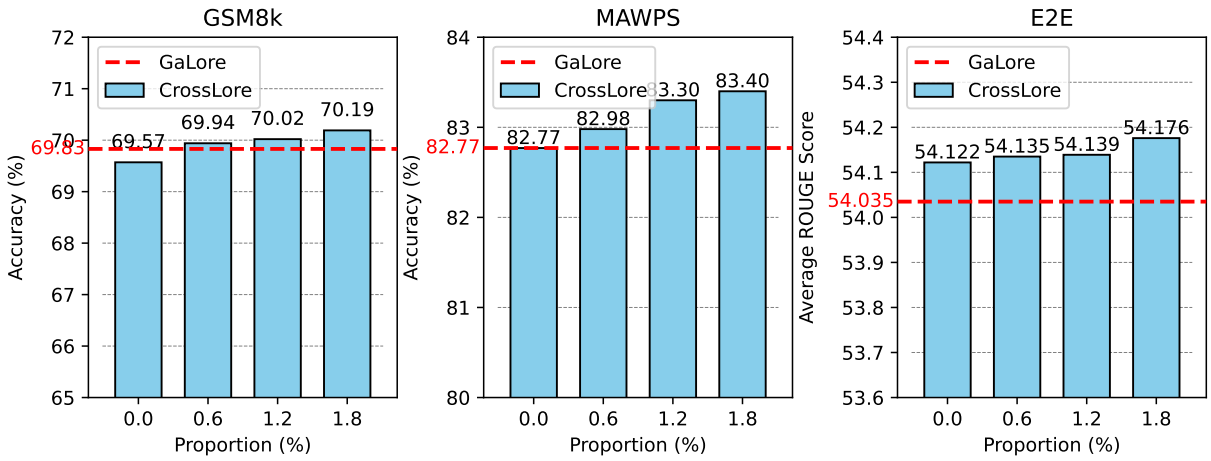


Figure 4: Ablation study on the sparsely coded residual, when fine-tuning LLaMA3-8B. The results are obtained by adjusting the proportion of non-zero elements in the sparse indexing matrix.

Methods	Rank	Time↓	Mem.↓	BLEU↑	NIST↑	MET↑	ROUGE-1/2/L↑	CIDEr↑
LoRA	16	2.03	28.41	62.50	4.54	35.73	68.53 / 36.89 / 45.05	1.37
DoRA	16	4.17	**28.21**	62.41	4.54	35.66	68.42 / 36.89 / 45.02	1.36
GaLore	16	11.29	28.52	65.14	4.82	37.13	70.80 / 40.50 / 48.64	1.74
CrossLore	16	2.75	28.22	**65.46**	**4.85**	**37.56**	**71.02 / 40.90 / 49.23**	**1.75**
LoRA	32	2.02	28.79	62.92	4.65	35.53	69.52 / 38.38 / 46.26	1.50
DoRA	32	4.17	28.58	62.87	4.64	35.44	69.38 / 38.30 / 46.20	1.50
GaLore	32	11.23	28.65	65.45	4.87	37.40	71.04 / 40.81 / 48.91	1.76
CrossLore	32	2.77	**28.33**	**65.55**	**4.89**	**37.59**	**71.18 / 41.11 / 49.18**	**1.78**
LoRA	128	1.98	30.08	64.90	4.77	37.05	70.67 / 40.08 / 48.18	1.67
DoRA	128	4.33	30.30	64.82	4.77	37.02	70.65 / 40.03 / 48.14	1.67
GaLore	128	11.41	29.64	64.79	5.08	36.96	71.13 / 41.34 / 49.34	1.80
CrossLore	128	2.97	**29.39**	**65.19**	**5.09**	**37.10**	**71.27 / 41.50 / 49.60**	**1.84**
LoRA	256	2.18	33.15	65.02	4.82	37.12	70.81 / 40.40 / 48.48	1.71
DoRA	256	4.42	33.41	65.07	4.83	37.19	70.90 / 40.47 / 48.61	1.72
GaLore	256	11.58	30.57	65.12	4.98	37.13	71.08 / 41.35 / 49.35	**1.82**
CrossLore	256	3.22	**30.20**	**65.19**	**5.05**	**37.28**	**71.29 / 41.49 / 49.53**	**1.82**

Table 8: Comparison of fine-tuning LLaMA2-13B with different PEFT methods on the E2E dataset. The metrics for fine-tuning time (Time) and memory consumption (Mem.) are 'hours' and 'GB'.

Models	Methods	CNN/DM				XSum			
		Time↓	R-1↑	R-2↑	R-L↑	Time↓	R-1↑	R-2↑	R-L↑
LLaMA2-7B	LoRA	0.37	31.68	11.45	21.85	0.25	37.25	13.46	29.50
	DoRA	0.68	31.98	11.57	22.01	0.50	37.52	13.53	29.91
	GaLore	1.18	33.64	12.98	**24.27**	1.04	40.24	16.07	32.92
	CrossLore	0.70	**33.76**	**13.02**	23.40	0.36	**40.36**	**16.25**	**33.00**
LLaMA2-13B	LoRA	0.51	32.77	12.14	22.98	0.35	39.15	15.11	31.53
	DoRA	0.94	32.69	12.00	23.10	0.71	39.38	15.40	31.72
	GaLore	2.23	33.65	13.06	24.33	2.03	42.54	**18.08**	35.15
	CrossLore	0.65	**34.05**	**13.25**	**24.44**	0.49	**42.70**	18.06	**35.26**

Table 9: Comparison of fine-tuning LLaMA2-7B and LLaMA2-13B with different PEFT methods at rank 32 on the CNN/DM and XSum datasets. The metrics for fine-tuning time is 'hours'. R-1, R-2, and R-L are abbreviations for ROUGE-1, ROUGE-2, and ROUGE-L, respectively.

Datasets	Models	0.0	0.6	1.2	1.8
GSM8k	LLaMA2-7B	33.64 (±0.97)	33.70 (±1.00)	33.80 (±0.69)	34.37 (±0.83)
	LLaMA3-8B	69.57 (±0.76)	69.94 (±0.83)	70.02 (±0.59)	70.19 (±0.54)
MAWPS	LLaMA2-7B	65.02 (±2.80)	65.13 (±0.51)	66.10 (±1.77)	66.18 (±1.34)
	LLaMA3-8B	82.77 (±0.59)	82.98 (±0.63)	83.30 (±1.00)	83.40 (±1.91)
E2E	LLaMA2-7B	53.966 (±0.113)	54.033 (±0.168)	54.035 (±0.042)	54.038 (±0.113)
	LLaMA3-8B	54.122 (±0.098)	54.135 (±0.077)	54.139 (±0.086)	54.176 (±0.080)

Table 10: Ablation study on the sparsely coded residual across different datasets, with proportions 0.0, 0.6, 1.2, and 1.8%.