


# HyC-LoRA: Memory Efficient LoRA Fine-tuning with Hybrid Activation Compression

**Yujin Wang** [1] **Shunan Dong** [1] **Zongle Huang** [1] **Yichen You** [1] **Liu He** [1] **Huazhong Yang** [1] **Yongpan Liu** [1] **Hongyang Jia** [1]

## Abstract

Large Language Models (LLMs) are widely used in applications like conversation and text summarization. With the demand for model customization and privacy, lightweight fine-tuning methods for large models have begun to receive widespread attention. Low-Rank Adaption (LoRA) is one of the most widely used fine-tuning algorithms, which significantly reduces the tunable weights and associated optimizer memory when transferring pre-trained LLMs to downstream tasks. However, past works lacked attention to the overhead of buffered activations in low-rank adaption, leading to suboptimal system memory usage.

To reduce buffered activation memory consumption and further enable the on-device memory-efficient fine-tuning system, we propose **HyC-LoRA**, a variant of the LoRA training method using a hybrid compression framework enabling almost 2-bit buffered activation quantization in all operators. HyC-LoRA observes that the temporarily buffered activation for backpropagation dominates the memory consumption in the LoRA fine-tuning process, and those in non-linear modules act as dominant memory consumers, whose quantization is more challenging. Based on this, HyC-LoRA proposes a hybrid compression mechanism with two tiers: **(1)** ***Intra-operator hybrid compression***: HyC-LoRA detects extreme outliers in buffered activation and mitigates the quantization error by structured outlier storage; **(2)** ***Inter-operator hybrid compression***: HyC-LoRA utilizes the LoRA adapter to achieve compensation for quantization errors and selective recomputation, through inter-operator reordering and fusion. Finally, HyC-LoRA implements a buffered activation compression system and integrates it with the existing machine learning framework to complete the last mile of lightweight storage for fine-tuning algorithms. Evaluations with multiple LLMs such as Llama series, in widely-used downstream tasks show the proposed HyC-LoRA framework achieves up to 3.97× end-to-end memory reduction compared to baseline, with negligible accuracy degradation. The code is available at `https://github.com/thu-ee-acts-lab/HyC-LoRA-release`.

## 1 Introduction

Large Language Models (LLMs) (Devlin et al., 2018; Liu et al., 2019; Touvron et al., 2023; Jiang et al., 2023) have achieved excellent performance in tasks such as text generation, article understanding, and more importantly, have shown excellent generalization capabilities. Supervised fine-tuning on pre-trained LLMs with task-specific data enhances downstream performance and customization (Wang et al., 2018; Cobbe et al., 2021). However, the training process introduces more unique memory components like ***optimizer state*** and ***buffered activation*** than the inference stage, and the high memory requirement increases the difficulty of its deployment on memory-constrained environments.

To reduce the hardware burden of large models, ***data compression*** methods such as quantization (Frantar et al., 2022; Xiao et al., 2023; Dettmers et al., 2022; Wei et al., 2022; Lin et al., 2023) and pruning (Sun et al., 2023; Frantar & Alistarh, 2023; Xia et al., 2023) are widely used for model deployment. Quantization compresses data by equally reducing the bit widths in a tensor, while pruning selectively retains the critical components. At the same time, a series of ***parameter-efficient fine-tuning*** (PEFT) methods (Ding et al., 2022) have been proposed to allow large models to efficiently gain the ability to handle downstream tasks without destroying their original generalization. LoRA (Hu et al., 2021) is one of the most preferred solutions for fine-tuning large models. Based on the assumption that the incremental update structure of weights is low-ranked after downstream fine-tuning, it freezes the backbone weight during fine-tuning and updates only a pair of low-rank matrices at the bypass, reducing the computation of backbone weight gradient and the storage of corresponding optimizer states. QLoRA (Dettmers et al., 2024) uses a combination of the

[1]Department of Electronic Engineering, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing 100084, China. Correspondence to: Hongyang Jia <hjia@tsinghua.edu.cn>.

*Proceedings of the $8^{th}$ MLSys Conference*, Santa Clara, CA, USA, 2025. 

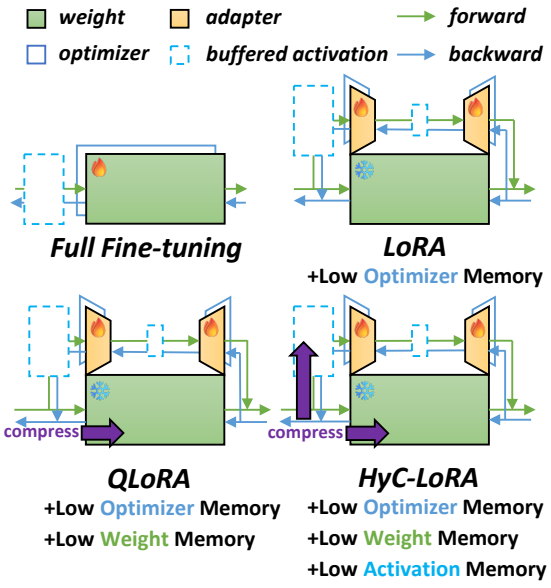



*Figure 1.* The memory requirements of different components (weight, optimizer, buffered activation) in different fine-tuning methods (Hu et al., 2021; Dettmers et al., 2024). HyC-LoRA achieves lightweight memory usage for all three components.

two methods proposed above to quantize the model backbone to 4-bit, further reducing the memory burden of the fine-tuning system.

However, the ***buffered activations*** that must be stored for backpropagation during the training phase remain a pressing problem. Some previous works have made attempts to compress buffered activation in small networks (Chen et al., 2021; Liu et al., 2022; Yu et al., 2024) or optimize for a particular operator (Dao et al., 2022; Zhang et al., 2023; Yang et al., 2024). Still, there is a lack of work that systematically explores how to achieve high-ratio buffered activation compression in the LoRA paradigm for large models to facilitate extreme compression at training time.

To address the above difficulties, we propose HyC-LoRA, a hybrid buffered activation compression framework for LoRA training (Figure 1). First, HyC-LoRA performs a detailed modeling of the buffered activation occupancy during LoRA fine-tuning. The modeling results show that the buffered activation occupancy in the LoRA training system does not decrease with the reduction of the tunable parameters, an important reason being that to maintain the integrity of the computational graphs, buffered activation in non-linear operators that are not directly related to updatable parameters still need to be stored, and some pre-experiments show that they are dominant in memory and more sensitive to compression. Then, HyC-LoRA addresses the above problems and proposes the following two strategies in different scales to enhance the compression effect of buffered activation: 1) ***Intra-operator hybrid compression***: By analyzing the data distribution of buffered activations in non-linear operators, we detect extreme outliers that are detrimental to quantization in some circumstances. These values tend to be distributed in fixed channels, so we propose a simple but effective structured outlier extraction to implement separate storage of quantized normal values and outliers with minimal overhead. 2) ***Inter-operator hybrid compression***: We observe that the non-linear operator's buffered activation is directly connected to the output of LoRA modules, so we multiplex the LoRA adapter to realize a hybrid data stream in the backpropagation phase and recompute the relevant modules at almost negligible cost, which not only reduces the loss of LoRA information due to compression but also reduces the storage of some buffered activations. Finally, we implement a buffered activation management system by integrating per-channel quantization with the aforementioned strategies, which achieves a minimum of 2-bit buffered activation compression in LoRA fine-tuning.

To demonstrate the effectiveness of our HyC-LoRA method, we conducted various experiments on different models (e.g., Llama (Touvron et al., 2023), Mistral (Jiang et al., 2023)), tasks (e.g., arithmetic reasoning (Cobbe et al., 2021), long sequence understanding (Azerbayev et al., 2024)), and scales (from 110M to 13B). To give a fair indication of the validity of our method, we compared it with other lightweight fine-tuning strategies to demonstrate its superiority. We also tested its memory gain and overhead on real hardware to prove its potential to enable lightweight fine-tuning. Experiment results show that HyC-LoRA can reduce buffered activation by up to about 8× and end-to-end memory by 1.57× to 3.97×, with minimal accuracy degradation and computation overhead.

In summary, our contributions can be listed as follows:

1. We conduct a comprehensive analysis of buffered activation source, breakdown, and distribution in the LoRA training system, pointing out that the buffered activations of non-linear operators are the bottleneck regarding memory usage and compression difficulty.
2. We propose HyC-LoRA, a lightweight LoRA variant that combines Intra-operator and Inter-operator hybrid compression with per-channel quantization to achieve a high buffered activation compression ratio.
3. We build a low-memory training system, which integrates HyC-LoRA with the existing machine learning framework and achieves a minimum of 2-bit buffered activation compression.
4. We evaluate HyC-LoRA at the algorithmic and system levels, showing that HyC-LoRA can reduce buffered activation by up to 8× and end-to-end memory by 1.57× to 3.97× with slight accuracy loss and better than other existing methods.

## 2 BACKGROUND

### 2.1 Quantization

Quantization (Nagel et al., 2021) is one of the most common data compression techniques for neural networks. Quantization can convert high bit-width floating point data $X$ to $q$-bit integer numbers $\widetilde{X}$. To realize the above transformation, we need to perform statistics on the data, based on which we calculate the scaling factor $s$ and zero point[1] $z$:

$$s = \frac{\max(X) - \min(X)}{2^q - 1}, z = -\lfloor \frac{\min(X)}{s} \rceil - 2^{q-1}. \quad (1)$$

Then the quantized value can be calculated as follows:

$$\widetilde{X} = \text{clamp}(\lfloor \frac{X}{s} + z \rceil, -2^{q-1}, 2^{q-1} - 1). \quad (2)$$

Notice that: **(1)** Neural network outputs typically exhibit consistent distributions across different inputs. Therefore, quantization parameters like scaling factors can be precomputed with a small calibration dataset and reused, eliminating the need for per-input computation. **(2)** The presence of outliers affects the calculation of statistics, making it challenging to find suitable scaling factors of the data and causing significant errors in subsequent quantization.

### 2.2 Fine-tuning Dataflow & Low Rank Adaption

Fine-tuning is a common approach to enhance the effectiveness of a pre-trained model's performance in specific domains, involving the computation of backpropagation and the updating of parameters, as exemplified by the full parameter fine-tuning of a linear layer:

$$\begin{aligned} &\text{forward}: \mathbf{Y} = \mathbf{XW}, \\ &\text{backward}: \frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbf{X}^T \frac{\partial \mathcal{L}}{\partial \mathbf{Y}}, \frac{\partial \mathcal{L}}{\partial \mathbf{X}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \mathbf{W}^T, \\ &\text{update}: \mathbf{W} \leftarrow \text{Adam}(\mathbf{W}, \mathcal{A}), \end{aligned} \quad (3)$$

where $\mathbf{W}$ is the weight parameter, $\mathbf{X}$ and $\mathbf{Y}$ are input/output activation, $\mathcal{A}$ represents corresponding optimizer state used for updating $\mathbf{W}$ (take Adam (Diederik, 2014) optimizer as example). Compared to inference, fine-tuning: **(1)** Input activations $\mathbf{X}$ should not be discarded immediately after forward propagation but should be buffered until the corresponding gradient has been computed after backpropagation. **(2)** Optimizer state $\mathcal{A}$ needs to store the gradient's first-order moments and second-order moments to achieve stable updates to the weights. These two components create an additional memory burden for fine-tuning.

A series of parameter-efficient fine-tuning (PEFT) methods have been proposed to mitigate the memory overhead of fine-tuning. Low-Rank Adaption (LoRA) (Hu et al., 2021) is the most widely used PEFT method for Transformer-based language models (Ding et al., 2022). Observing that a pair of low-rank matrices can approximate the change in weights after fine-tuning, LoRA adds a new tunable pair of adapter matrices bypassing the main weight:

$$\mathbf{Y} = \underbrace{\mathbf{XW}}_{\text{main weight}} + \underbrace{\mathbf{XAB}}_{\text{adapter}}, \quad (4)$$

where $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ is the main weight with hidden dimension $d_1$ and $d_2$, and $\mathbf{A} \in \mathbb{R}^{d_1 \times r}, \mathbf{B} \in \mathbb{R}^{r \times d_2}$ are low rank adapters with LoRA rank size $r$. During LoRA fine-tuning, the main weight $\mathbf{W}$ is frozen, and only $\mathbf{A}$ and $\mathbf{B}$ matrices are updateable. The method can significantly reduce the optimizer-state parameters during training and checkpoint size after training since $r << d$.

## 3 OBSERVATIONS OF LORA FINE-TUNING SYSTEM

In this section, we first analyze the memory components of the LoRA fine-tuning process and point out that buffered activation is a critical bottleneck in the training system. We also note that buffered activation of non-linear operators has a higher proportion, and direct compression degrades training accuracy, which poses a challenge to implementing memory-efficient training systems.

Figure 2(a) shows the breakdown of each component in QLoRA fine-tuning (Dettmers et al., 2024) under different training configurations, and it is evident that buffered activation becomes predominant and exhibits a rapid increase as sequence length and batch size grow. To explore the source of buffered activation clearly, we analyzed the backpropagation dataflow and enumerated all the buffered activations in Figure 3 **Left**. Depending on their computational properties, we classify the buffered activations into two categories:

**(1)** *Buffered activation for linear modules (linear buffered activation).* The backpropagation dataflow of LoRA adapters can be formulated as follows:

$$\begin{aligned} &\mathbf{Y} = \mathbf{X}(\mathbf{W} + \mathbf{AB}) \Rightarrow \\ &\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \mathbf{X}^T \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \mathbf{B}^T, \frac{\partial \mathcal{L}}{\partial \mathbf{B}} = (\mathbf{XA})^T \frac{\partial \mathcal{L}}{\partial \mathbf{Y}}, \\ &\frac{\partial \mathcal{L}}{\partial \mathbf{X}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} (\mathbf{W}^T + \mathbf{B}^T \mathbf{A}^T). \end{aligned} \quad (5)$$

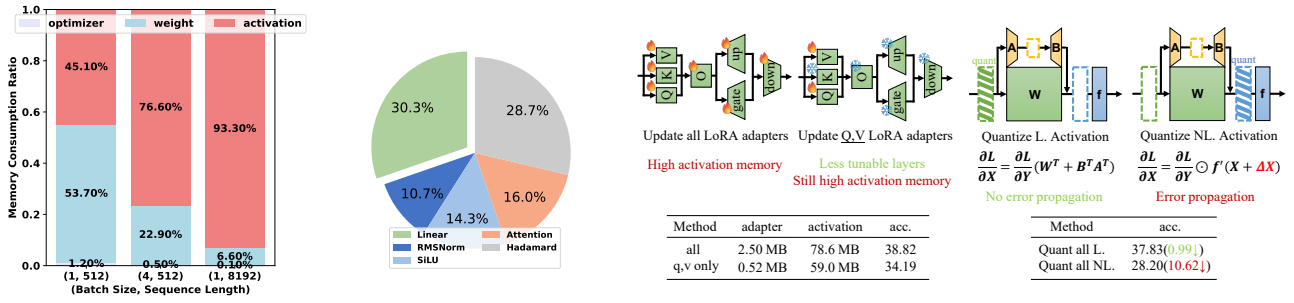
The buffered activation $\mathbf{X}$ and $(\mathbf{XA})$ are only relevant for the current layer's parameter update and are not involved in the computation of the previous layer's gradient.

**(2)** *Buffered activation for non-linear modules (non-linear buffered activation).* The non-linear operators $f$ involved in the Transformer layer include RMSNorm (Zhang & Sennrich, 2019), Attention (Vaswani et al., 2017), SiLU (Elfwing et al., 2017), etc., and their backpropagation dataflow can be represented as:

$$\mathbf{Y} = f(\mathbf{X}) \Rightarrow \frac{\partial \mathcal{L}}{\partial \mathbf{X}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \odot f'(\mathbf{X}). \quad (6)$$

Even though no updatable parameters exist or are frozen, the computation of the previous layer's gradients still relies on the buffered activation and cannot be dropped.

[1] for asymmetric quantization only




| Method | adapter | activation | acc. |
|---|---|---|---|
| all | 2.50 MB | 78.6 MB | 38.82 |
| q,v only | 0.52 MB | 59.0 MB | 34.19 |

| Method | acc. |
|---|---|
| Quant all L. | 37.83(0.99↓) |
| Quant all NL. | 28.20(10.62↓) |

(a) (b) (c) (d)

*Figure 2.* **(a)** Theoretical (same below) memory breakdown of weight, optimizer and buffered activation of Llama-2-7B in QLoRA training; **(b)** Percentage of buffered activations from different operators in Llama-2-7B; **(c)** Memory of LoRA adapter and buffered activation in one Llama-2-7B layer, with different fine-tuning strategies; **(d)** The accuracy degradation caused by compressing non-linear buffered activation is more pronounced than compressing linear buffered activation.

The presence of buffered activation poses the following challenges for the design of lightweight algorithms:

**(1)** *Non-linear buffered activation is hard to eliminate.* As shown in Figure 2(b), the percentage of non-linear buffered activation in Llama-2-7B reaches 69.7%. Figure 2(c) shows a common approach (Hu et al., 2021) to reduce the tunable parameters in LoRA, i.e., only performing LoRA fine-tuning to the Q, V weight matrix, which reduces the tunable parameters by 79.2%. However, since non-linear buffered activation must be preserved to maintain the computational graph's integrity, the total buffered activation's reduction is only 24.9%. Qualitatively speaking, in the LoRA algorithm, buffered activation memory size is not entirely correlated with the size of the tunable parameters, the nature of the computational graph must also be considered.

**(2)** *Non-linear buffered activation is hard to compress.* The above two kinds of buffered activations have different computational properties in backpropagation. As shown in Figure 2(d), the compression error of linear buffered activation does not propagate to the preceding module, whereas non-linear buffered activation passes to the preceding module and accumulates. The experiment confirms this assumption: if we quantize all the linear buffered activations to 2-bit, the drop of accuracy is negligible (38.82% $\rightarrow$ 37.83%), while quantizing the non-linear buffered activations causes a significant drop (38.82% $\rightarrow$ 28.20%).

In summary, the compression of buffered activations is a crucial factor driving further light-weighting of LoRA fine-tuning, and handling non-linear buffered activations is significant. In the following sections, we will show how HyC-LoRA enables more memory-efficient fine-tuning while ensuring the accuracy of fine-tuning results.

# 4 HyC-LoRA

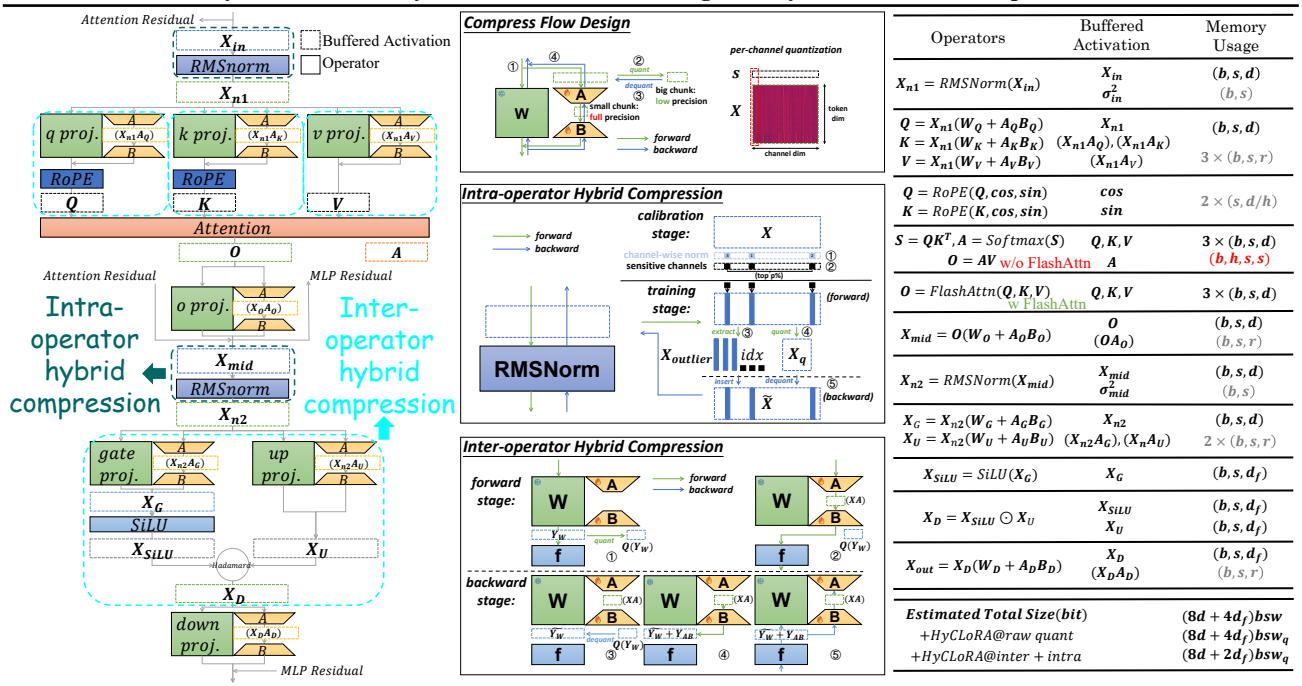
In contrast to conventional compression methods that rely on a single technique, our proposed method, HyC-LoRA, effectively reconstructs buffered activations using hybrid compression. We not only incorporate the distributional characteristics and computational data flow of buffered activations to reduce the compression error but also implement the relevant operators efficiently to minimize the additional overhead.

## 4.1 Compress Flow Design

Previous works (Xiao et al., 2023; Lin et al., 2023) show that the activations exhibit per-channel distribution, and per-tensor quantization causes significant performance dropping. To achieve a balance between compression ratio and error, we use ***per-channel quantization*** for most buffered activations (marked in **black** of Figure 3 Right).

Specifically, as shown in Figure 3 **Middle**, exploiting that model activation features do not change much during LoRA fine-tuning, the whole process can be divided into *calibration stage* and *training stage* to reduce statistical overhead. During the calibration stage, we obtain buffered activations using several calibration data samples and compute the scaling factor $\mathbf{s}$ per channel. These scaling factors reflect the distributional characteristics of the activation stably, thus eliminating the need for recalculations in the training stage. The number of calibration samples is typically small, so the additional compute overhead introduced is almost negligible. During the training stage, we directly use the per-channel scaling factor $\mathbf{s}$ to quantize the buffered activations and get the low bit representation for save in forward propagation (❶, ❷); then, they are dequantized and participate in subsequent backward (❸, ❹).

There are two kinds of exceptional cases: **(1)** For the attention map in naive attention (marked in red, not reserved in FlashAttention (Dao, 2023) implementation), we adopt the pruning method in (Jiang et al., 2022) because it naturally has high sparsity, and its sparse pattern is not fixed. **(2)** For the small activation chunks (marked in grey), we still leave them at full precision because their share of total memory is negligible, and it is also necessary for retaining little but




| Operators | Buffered Activation | Memory Usage |
|---|---|---|
| $X_{n1} = RMSNorm(X_{in})$ | $X_{in}$<br>$\sigma^2_{in}$ | $(b,s,d)$<br>$(b,s)$ |
| $Q = X_{n1}(W_Q + A_Q B_Q)$<br>$K = X_{n1}(W_K + A_K B_K)$<br>$V = X_{n1}(W_V + A_V B_V)$ | $X_{n1}$<br>$(X_{n1}A_Q), (X_{n1}A_K)$<br>$(X_{n1}A_V)$ | $(b,s,d)$<br>$3 \times (b,s,r)$ |
| $Q = RoPE(Q, cos, sin)$<br>$K = RoPE(K, cos, sin)$ | $cos$<br>$sin$ | $2 \times (s, d/h)$ |
| $S = QK^T, A = Softmax(S)$<br>$O = AV$ w/o FlashAttn | $Q, K, V$<br>$A$ | $3 \times (b,s,d)$<br>$(b,h,s,s)$ |
| $O = FlashAttn(Q, K, V)$ w FlashAttn | $Q, K, V$ | $3 \times (b,s,d)$ |
| $X_{mid} = O(W_O + A_O B_O)$ | $O$<br>$(OA_O)$ | $(b,s,d)$<br>$(b,s,r)$ |
| $X_{n2} = RMSNorm(X_{mid})$ | $X_{mid}$<br>$\sigma^2_{mid}$ | $(b,s,d)$<br>$(b,s)$ |
| $X_G = X_{n2}(W_G + A_G B_G)$<br>$X_U = X_{n2}(W_U + A_U B_U)$ | $X_{n2}$<br>$(X_{n2}A_G), (X_n A_U)$ | $(b,s,d)$<br>$2 \times (b,s,r)$ |
| $X_{SiLU} = SiLU(X_G)$ | $X_G$ | $(b,s,d_f)$ |
| $X_D = X_{SiLU} \odot X_U$ | $X_{SiLU}$<br>$X_U$ | $(b,s,d_f)$<br>$(b,s,d_f)$ |
| $X_{out} = X_D(W_D + A_D B_D)$ | $X_D$<br>$(X_D A_D)$ | $(b,s,d_f)$<br>$(b,s,r)$ |
| ***Estimated Total Size(bit)*** | | $(8d + 4d_f)bsw$ |
| $+HyCLoRA@raw\ quant$ | | $(8d + 4d_f)bsw_q$ |
| $+HyCLoRA@inter + intra$ | | $(8d + 2d_f)bsw_q$ |

*Figure 3.* **Left**: Overview compute flow of LoRA training (Llama-2-7B Transformer block). The figure also specifies the location of buffered activation and indicates the scopes of the two hybrid mechanisms. We also show RoBERTa block in Appendix (Figure 10 **Left**). **Middle**: Dataflow of overall compress flow design, intra-operator, and inter-operator hybrid compression. **Right**: Modeling of buffered activations during training (take Llama-2-7B as example). We enumerate *shape* (marked in "()") in related operators. Operators that do not need buffered activations, like residual connection and reshaping, are not listed for simplicity. Some small chunks of buffered activation have a negligible share of memory, so we do not compress them and they are marked grey in the table. $d$: hidden dimension; $d_f$: FFN's hidden dimension; $s$: training sequence length; $h$: head dimension; $w$: original bit-width of buffered activation; $w_q$: quantization bit-width.

important activation information.

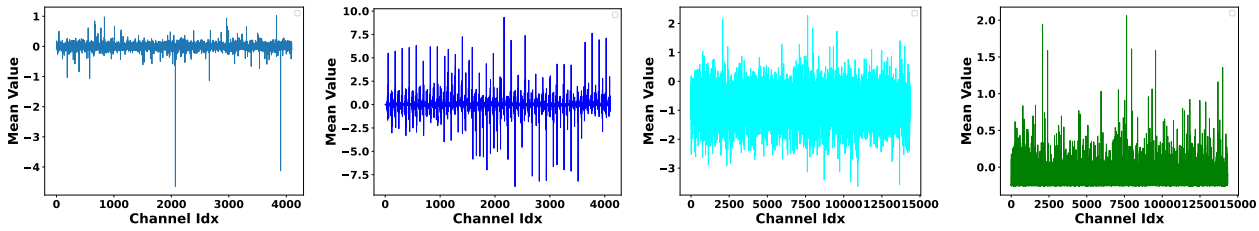
### 4.2 Intra-operator Hybrid Compression

Prior studies on LLM inference (Xiao et al., 2023; Lin et al., 2023; Zhao et al., 2024b) have pointed out that the presence of outliers is a significant challenge to maintaining quantization accuracy. However, these works focus on solving the compression problems of linear operators, and there is a lack of exploration of the relevant properties of non-linear operators. We analyze the distribution of non-linear buffered activations as shown in Figure 4. We capture extreme outliers in the RMSNorm's buffered activation, which are only distributed in very few channels (<1%). In **Q** buffer and **K** buffer of Attention, activation appears to have a regular pattern of outliers, mainly related to the positional coding module introduced by the RoPE (Su et al., 2024) module. In other modules like SiLU, we do not observe a tangible pattern of outliers.

Although outlier extraction can significantly mitigate the quantization error, it may involve complex sorting operators and irregular memory accesses. Luckily, for the RMSNorm operator, the outliers only appear in specific channels. To mitigate the overhead of the associated operations, we propose a two-stage approach called **Structured Outlier Extraction** to deal with outliers, as shown in Figure 3 **Middle**. In the *calibration stage*, we calculate the buffered activation's per-channel norm $\mathbf{n} \in \mathbb{R}^{1\times d}$ (❶), then select max $p_n\%$ as the *sensitive channels* and mark their index (❷). In the *training stage*, we retrieve and propose these fixed channels, set them to full precision as $\mathbf{X}_{outlier} \in \mathbb{R}_{s\times(d\times p_n\%)}$ (❸) and quantize the remainder as $\mathbf{X}_q \in \mathbb{R}_{s\times d}$ (❹) in the forward process, then during backward process, the buffered activations $\widetilde{\mathbf{X}} \in \mathbb{R}_{s\times d}$ are merged from the two parts (❺). The full computing process is shown in Algorithm 1.

In the training process, our approach significantly reduces computational overhead by selectively accessing and modifying only a minimal number of columns, rather than exhaustively searching for outliers across the entire buffered activation, thereby ensuring negligible additional computational burden. Moreover, the percentage of quantization-sensitive channels is very low, e.g., the average quantization bit is only $16 \times 0.005 + 2 = 2.08$ bits under the condition of selecting 0.5% channels, which can be considered as no change in compression ratio.

Additionally, for the **Q**,**K** buffer in Attention module, to avoid the corruption of the data distribution, we do not quantize the post-RoPE value directly; instead, we quantize the pre-RoPE value during forward propagation, then de-




(a) RMSNorm ($\mathbf{X}_{in}$) (b) Attention-Q ($\mathbf{Q}$) (c) SiLU ($\mathbf{X}_G$) (d) Hadamard ($\mathbf{X}_{SiLU}$)

*Figure 4.* Mean values per channel of different non-linear buffered activations from Mistral-7B. We detected the presence of extreme outliers in the buffered activation of the RMSNorm module. We also detected particular distribution patterns in the buffered activation of attention due to the RoPE operation.

**Algorithm 1** Structured Outlier Extraction

**Require:** input activation $\mathbf{X} \in \mathbb{R}^{s\times d}$, input gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}$, RMSNorm/LayerNorm operator XNorm

***calibration stage:***
  compute per-channel norm: $\mathbf{n} = ||\mathbf{X}_{:,j}||_{j=1}^{d}$
  select outlier index: $id = \text{argmax}(\mathbf{n}, d \times p_n\%)$
  compute scaling factor: $\mathbf{s} = \text{scale}(\mathbf{X}_{[0,d]\setminus id})$
***training stage (forward):***
  forward: $\mathbf{Y} = \text{XNorm}(\mathbf{X})$
  select outlier part: $\mathbf{X}_{outlier} = \mathbf{X}_{:,j;j\in id}$
  compress rest part: $\mathbf{X}_q = \text{Quant}(\mathbf{X} - \mathbf{X}_{outlier})$
  save: $\mathbf{X}_q, \mathbf{X}_{outlier}$
***training stage (backward):***
  load: $\mathbf{X}_q, \mathbf{X}_{outlier}$
  decompress: $\widetilde{\mathbf{X}} = \text{Dequant}(\mathbf{X}_q) + \mathbf{X}_{outlier}$
  backward: $\frac{\partial \mathcal{L}}{\partial \mathbf{X}} = \text{XNorm}'(\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}, \widetilde{\mathbf{X}})$

**Algorithm 2** LoRA Reorder Computing

**Require:** backbone weight $\mathbf{W} \in \mathbb{R}^{d_1\times d_2}$, LoRA adapter weight $\mathbf{A} \in \mathbb{R}^{d_1\times r}$, $\mathbf{B} \in \mathbb{R}^{r\times d_2}$, input activation $\mathbf{X} \in \mathbb{R}^{s\times d_1}$, non-linear operator $f$, non-linear operator gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{O}} \in \mathbb{R}^{s\times d_2}$

***forward stage:***
  forward: $\mathbf{Y}_W = \mathbf{XW}$,$\mathbf{Y}_{AB} = (\mathbf{XA})\mathbf{B}$
  quantize: $\mathbf{Y}_{W,q} \leftarrow \text{Quant}(\mathbf{Y}_W)$
  save: $\mathbf{Y}_{W,q}, (\mathbf{XA})$
  merge: $\mathbf{Y} = \mathbf{Y}_W + \mathbf{Y}_{AB}$
***backward stage:***
  load: $\mathbf{Y}_{W,q}, (\mathbf{XA})$
  dequantize: $\widetilde{\mathbf{Y}_W} \leftarrow \text{Dequant}(\mathbf{Y}_{W,q})$
  recompute: $\mathbf{Y}_{AB} = (\mathbf{XA})\mathbf{B}$, $\widetilde{\mathbf{Y}} = \widetilde{\mathbf{Y}_W} + \mathbf{Y}_{AB}$
  backward: $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}} = \frac{\partial \mathcal{L}}{\partial \mathbf{O}} \odot f'(\widetilde{\mathbf{Y}})$

quantize it and apply RoPE during backward for follow-up calculation.

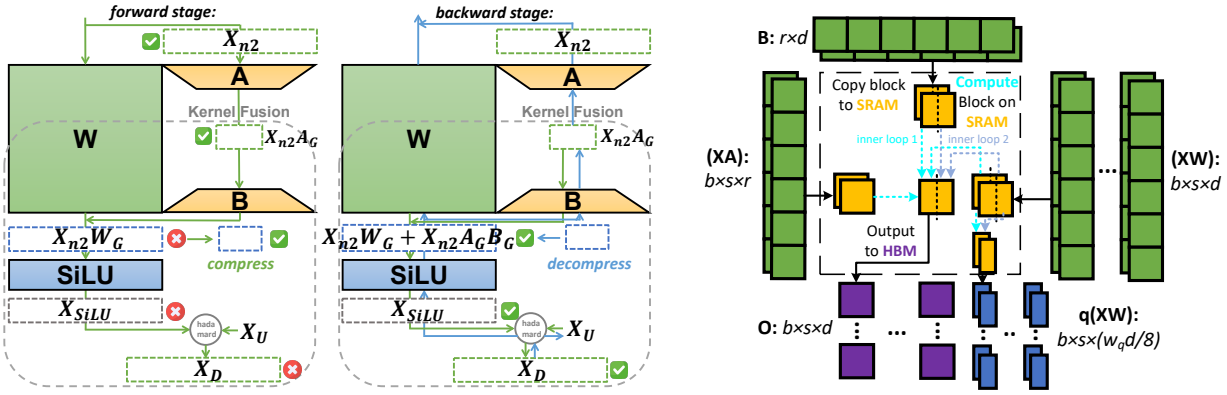
### 4.3 Inter-operator Hybrid Compression

Although no apparent outlier features are captured in other non-linear buffered activations, we can still analyze the sources of their quantization errors at the inter-operator compute flow level. We observe that the buffered activation of non-linear operators $\mathbf{Y}$ is directly connected to the output of the linear layer, which information is derived from two parts: the output $\mathbf{Y}_W$ from the linear backbone, containing the embedded knowledge of original model, and the output $\mathbf{Y}_{AB}$ from the LoRA adapters, containing the knowledge learned during the fine-tuning process. The buffered activations of non-linear operators are essentially the aggregate of these two components, and the direct quantization would result in the activation information, especially the newly learned representation in the LoRA adapter, being corrupted by the compression process.

As shown in Figure 3 **Middle**, in the LoRA adapter module, we buffer the output of LoRA-A adapter $(\mathbf{XA}) \in \mathbb{R}^{s\times r}$ to calculate the gradient of LoRA-B. The LoRA-A adapter performs a low-rank projection of activation ($r << d/d_f$), so the size of $(\mathbf{XA})$ is much smaller than the output $\mathbf{Y} \in \mathbb{R}^{s\times d}$, with negligible memory footprint in the system even if saving it in full precision. This heterogeneous accuracy setting provides an opportunity for non-destructive reconstruction of the LoRA adapter's output information.

Inspired by this, we propose **LoRA Reorder Computing**, which performs a hybrid rearrangement of the operator's logical order in the forward and backward-propagation phases. During the forward stage, only buffered activations from the main part $\mathbf{Y}_W$ is quantized to $\mathbf{Y}_{W,Q}$ (❶), while the buffered activation's information of the LoRA adapter is retained in $(\mathbf{XA})$ (❷). During the backward stage, the complete LoRA representation $\mathbf{Y}_{AB}$ will be obtained by multiplying $(\mathbf{XA})$ by $\mathbf{B}$ and adding it to the dequantized value of the main output $\widetilde{\mathbf{Y}_W}$ as $\widetilde{\mathbf{Y}}$ (❸, ❹), then the reconstructed value is used for non-linear operator's backpropagation (❺). This method reuses the adapter structure and buffered activation in LoRA, so no additional memory is required. The computation flops of its extra operation $(\mathbf{XA})\mathbf{B} + \widetilde{\mathbf{Y}_W}$ is much smaller than that of full forward and backward propagation so the additional computation overhead is negligible. The full computing process is shown in Algorithm 2.

As shown in Figure 5, to take full advantage of inter-operator




(a) (b)

*Figure 5.* (a) Scope of operator fusion for inter-operator hybrid compression in Llama-2, The $\mathbf{X}_{SiLU}$ and $\mathbf{X}_D$ activation are dropped after forward propagation, and are recomputed with the fused LoRA compensation module, reducing the buffered activation storage. The case for RoBERTa is shown in Appendix (Figure 11). (b) Schematic of the operator fusion mechanism (take forward propagation as example), which significantly reduces data transfer.

optimization, we write kernels to fuse LoRA reorder computing with compression/decompression process and element-wise operations (e.g., SiLU and Hadamard product), which can be used as epilogue operations at almost zero cost. Although the idea of kernel fusion is covered in previous works (Daniel Han & team, 2023; Ansel et al., 2024; Hong et al., 2024), in our memory efficient fine-tuning scenario, there are at least two unique benefits: **(1)** The element-wise operator's buffered activation can be dropped during forward and re-calculated as epilogue result at almost nil cost, which not only further reduces buffered activation memory usage but also consequently reduces the overhead of compression/decompression. **(2)** The fused operator requires only a single read and write to the off-chip tensor, significantly reducing the EMA (external memory access) and offsetting the small additional overhead of LoRA reorder computing.

## 5 Experiment

### 5.1 Setup

**Models.** To demonstrate the algorithmic effectiveness of our approach, we conduct experiments on commonly used decoder-based model families (e.g., Llama (Touvron et al., 2023), Mistral (Jiang et al., 2023)). To illustrate that our approach can be extended to different model architectures, we likewise experimented on the encoder-based models like RoBERTa (Liu et al., 2019). By default, the LoRA adapters are enabled in every linear layer in the model except the classification head. The LoRA rank size $r$ is set to 16 and the LoRA scaling factor $\alpha$ is set to 1. The model parameters involved in the experiment are shown in Appendix (Table 7).

**Tasks.** For decoder-based large language models, we perform: **(1)** Single-task fine-tuning on the GSM8K (Cobbe et al., 2021) and Wikitext-2 (Merity et al., 2016) datasets following (Li et al., 2023) to test the fine-tuned model's capabilities on arithmetic reasoning and text understanding. **(2)** General-task fine-tuning on the Math10K (Hu et al., 2023) dataset following (Liao & Monz, 2024) to test the model's ability to generalize after model fine-tuning. **(3)** Long-sentence fine-tuning on the Redpajama (Computer, 2023) dataset following (Chen et al., 2023) to test the model's ability on long sequence understanding, whose training scenario particularly highlights the system-level advantages of our approach, as the memory footprint of buffered activations becomes more substantial in long-sequence training contexts. For RoBERTa-Base, we fine-tune and evaluate the models on General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) following (Hu et al., 2021) to test natural language understanding ability. We list some essential training settings of different tasks in Appendix (Table 8).

**Implementation Details.** We implemented our algorithm using PyTorch framework (Paszke et al., 2019) and customized the backward algorithm of all the operators involved in the standard Transformer framework with Triton language (Lang, 2023), some kernels like RoPE (Su et al., 2024) draw on the implementation in unsloth (Daniel Han & team, 2023). We freeze all the backbone weights and quantize them to Nfloat4 format as (Dettmers et al., 2024), while the compute precision is set to bfloat16. The calibration step is set to 5. The RMSNorm outlier channel ratio $p_n$ is set to $0.5\%$. Scaled dot product attention (Vaswani et al., 2017) is used in fine-tuning by default, and the attention sparse ratio $p_a$ is set to $5\%$, while for long-sentence task fine-tuning, we apply FlashAttention (Dao et al., 2022) for memory and computation efficiency.

**Compared methods.** We set QLoRA (Dettmers et al., 2024)

as our *baseline*, and we compare HyC-LoRA with different lightweight variants of QLoRA. The consideration for this setting is that QLoRA has fixed the quantization of weight to 4-bit, and the additional number of parameters added by the LoRA adapter and the corresponding optimizer state is negligible in the system, so the difference in memory consumption will mainly depend on the usage of buffered activation. We list these methods as follows: **(1)** QST (Zhang et al., 2024b): QST adds a small separate network beside the backbone-quantized network. Since the backpropagation only needs to go through the small separate network, its buffered activation can be significantly reduced. **(2)** SparseBP (Zhu et al., 2023): SparseBP eliminates the preceding layers' buffered activations by blocking the backpropagation graph. To control the memory usage of the weight and optimizer in line with baseline experiments, instead of performing the full parameter fine-tuning on the partial network in the original paper, we perform QLoRA fine-tuning on partial Transformer layers, and the fine-tuned linear layers include q, k and v projection layer and gate, up projection, aligning with the original paper's experimental setup for Transformer architectures. **(3)** BackRazor (Jiang et al., 2022): BackRazor proposes an asymmetric pruning strategy in which buffered activations are pruned unstructurally and stored in the sparse form to be recovered in the backpropagation phase. **(4)** LoRA-FA (Zhang et al., 2023): LoRA-FA freezes the LoRA-A adapter and only fine-tunes the LoRA-B adapter to reduce the linear layer's buffered activation. The theoretical buffered activation reduction ratio of these methods is shown in Appendix (Table 9 and 10).

Additionally, we have not listed some more lightweight fine-tuning methods (e.g., bias-tuning (Zaken et al., 2022)) because they do not perform well on large models. We also do not list previous pure-quantization frameworks for small models like ActNN (Chen et al., 2021) and GACT (Liu et al., 2022) because of their incompatibility with large model training systems. Therefore, we reimplemented the quantization method and used a customized setup.

### 5.2 Accuracy Evaluation

Table 1 presents the model's zero-shot accuracy on GSM8K and perplexity on Wikitext-2. In HyC-LoRA@4-bit+intra+inter setting, accuracy loss on GSM8K is minimal (1.45%, 0.91%, 1.44%, 1.06% across models). For 2-bit circumstances, HyC-LoRA outperforms the baseline more significantly; for Llama-2-7B, it achieves a 6.37% relative improvement (29.49% → 35.86%). The other memory-efficient methods have a gap in algorithm performance compared to HyC-LoRA, take Wikitext-2 dataset as an example, compared to SparseBP (1.17 perplexity increase with about 60% activation reduction), HyC-LoRA@4-bit+intra+inter achieves about 75% reduction with only a 0.02 perplexity increase.

*Table 1.* Algorithm performance on GSM8K (Cobbe et al., 2021) (represented by "G") and WikiText-2 (Merity et al., 2016) (represented by "W"). We report zero-shot accuracy for GSM8K (the larger the better) and perplexity (the smaller the better). We mark "intra" as *Intra-operator Hybrid Compression* and *Inter-operator Hybrid Compression*, the best results under the same compression ratio are **bolded** (the same as below).

| Method | Quant Bit | Intra | Inter | TinyLlama-1.1B G↑ | TinyLlama-1.1B W↓ | Llama-2-7B G↑ | Llama-2-7B W↓ | Llama-2-13B G↑ | Llama-2-13B W↓ | Mistral-7B G↑ | Mistral-7B W↓ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Baseline | 16 | - | - | 15.85 | 8.24 | 38.82 | 5.51 | 48.75 | 5.02 | 53.68 | 5.33 |
| SparseBP@0.5[a] | 16 | - | - | 8.57 | 9.28 | 16.76 | 7.15 | 21.00 | 6.74 | 36.09 | 5.71 |
| BackRazor@0.2[b] | 16 | - | - | 13.12 | 8.42 | 33.28 | 5.68 | 42.46 | 5.06 | 44.88 | 5.39 |
| QST | 16 | - | - | 2.12 | 13.44 | 7.51 | 8.98 | 12.81 | 10.73 | - | - |
| LoRA-FA | 16 | - | - | 9.78 | 8.98 | 32.52 | 6.77 | 43.67 | 6.02 | 54.51 | 5.48 |
| HyC-LoRA | 4 | ✗ | ✗ | 14.10 | 8.28 | 35.86 | 5.71 | 37.53 | 5.02 | 52.16 | 5.36 |
| | 4 | ✓ | ✗ | **14.40** | 8.27 | 37.23 | 5.58 | 45.79 | 5.02 | **52.62** | **5.35** |
| | 4 | ✓ | ✓ | 14.03 | **8.25** | **37.91** | **5.57** | **47.31** | **5.01** | 51.63 | 5.36 |
| | 2 | ✗ | ✗ | 11.30 | 8.39 | 29.49 | 5.81 | 42.38 | 5.06 | 45.94 | 5.46 |
| | 2 | ✓ | ✗ | 11.90 | 8.37 | 34.65 | **5.76** | 43.90 | 5.05 | 46.02 | 5.43 |
| | 2 | ✓ | ✓ | **12.96** | **8.32** | **35.86** | 5.82 | **44.05** | **5.04** | **47.16** | **5.42** |

*Table 2.* Zero-shot accuracy performance on four arithmetic reasoning tasks (GSM8K (Cobbe et al., 2021), SVAMP (Patel et al., 2021), mawps (Koncel-Kedziorski et al., 2016) and AQuA (Ling et al., 2017)) after fine-tuning on Math10K (Hu et al., 2023).

| Method | Quant Bit | Intra | Inter | GSM8K | SVAMP | mawps | AQuA | Avg. |
|---|---|---|---|---|---|---|---|---|
| **TinyLlama-1.1B** | | | | | | | | |
| Baseline | 16 | - | - | 11.45 | 28.8 | 67.64 | 20.08 | 31.99 |
| SparseBP@0.5 | 16 | - | - | 5.84 | 21.8 | 55.88 | 23.22 | 26.68 |
| BackRazor@0.2 | 16 | - | - | 7.20 | 26.5 | 63.02 | 17.72 | 28.61 |
| LoRA-FA | 16 | - | - | 5.91 | 20.4 | 50.42 | 25.20 | 25.48 |
| HyC-LoRA | 4 | ✗ | ✗ | 8.64 | 26.1 | 62.61 | 22.44 | 29.94 |
| | 4 | ✓ | ✗ | 8.04 | 26.5 | 65.97 | **22.44** | 30.74 |
| | 4 | ✓ | ✓ | **9.55** | **27.9** | **68.06** | 21.65 | **31.79** |
| | 2 | ✗ | ✗ | 8.04 | 22.6 | 55.04 | 20.08 | 26.44 |
| | 2 | ✓ | ✗ | 8.19 | 21.4 | 57.14 | **25.98** | 28.18 |
| | 2 | ✓ | ✓ | **8.34** | **24.5** | **59.24** | 23.62 | **28.93** |
| **Llama2-7B** | | | | | | | | |
| Baseline | 16 | - | - | 43.21 | 58.5 | 84.45 | 24.80 | 52.74 |
| SparseBP@0.5 | 16 | - | - | 19.48 | 38.9 | 75.63 | 20.87 | 38.72 |
| BackRazor@0.2 | 16 | - | - | 39.19 | 55.8 | 86.14 | 23.22 | 51.09 |
| LoRA-FA | 16 | - | - | 33.74 | 51.5 | 79.83 | 20.08 | 46.29 |
| HyC-LoRA | 4 | ✗ | ✗ | 39.50 | **57.3** | 83.19 | 21.26 | 50.31 |
| | 4 | ✓ | ✗ | **40.94** | 56.4 | **84.03** | **25.98** | **51.84** |
| | 4 | ✓ | ✓ | 39.50 | 56.6 | 82.35 | 22.44 | 50.22 |
| | 2 | ✗ | ✗ | 37.07 | 53.7 | 81.51 | **25.98** | 49.57 |
| | 2 | ✓ | ✗ | 37.07 | 52.0 | 81.51 | 25.20 | 48.95 |
| | 2 | ✓ | ✓ | **38.74** | **54.7** | **82.77** | 22.04 | **49.57** |
| **Mistral-7B** | | | | | | | | |
| Baseline | 16 | - | - | 59.89 | 71.7 | 89.08 | 27.95 | 62.16 |
| SparseBP@0.5 | 16 | - | - | 48.05 | 64.7 | 88.23 | 28.02 | 57.25 |
| BackRazor@0.2 | 16 | - | - | 58.07 | 70.8 | 89.50 | 27.56 | 61.48 |
| LoRA-FA | 16 | - | - | 60.88 | 71.6 | 90.34 | 28.74 | 62.89 |
| HyC-LoRA | 4 | ✗ | ✗ | 58.83 | 69.6 | 88.66 | 27.95 | 61.26 |
| | 4 | ✓ | ✗ | 59.74 | 68.3 | 88.66 | 27.56 | 61.06 |
| | 4 | ✓ | ✓ | **59.89** | **72.4** | **90.33** | **29.13** | **62.94** |
| | 2 | ✗ | ✗ | 55.34 | 66.7 | 86.55 | 25.59 | 58.55 |
| | 2 | ✓ | ✗ | 56.94 | **68.8** | 86.97 | 26.38 | 59.77 |
| | 2 | ✓ | ✓ | **57.46** | 67.0 | **87.82** | **27.95** | **60.06** |

*Table 3.* Perplexity on proof-pile (Azerbayev et al., 2024) and PG19-val (Rae et al., 2019) datasets after fine-tuning on Redpajama (Computer, 2023) dataset.

| Method | Quant Bit | Intra | Inter | Llama-2-7B proof-pile | Llama-2-7B PG19-val | Llama-2-13B proof-pile | Llama-2-13B PG19-val |
|---|---|---|---|---|---|---|---|
| No Fine-tune | - | - | - | 4.373 | 18.060 | 2.739 | 8.104 |
| Baseline | 16 | - | - | 2.833 | 8.380 | 2.616 | 7.185 |
| HyC-LoRA | 4 | ✗ | ✗ | 2.846 | 8.451 | 2.626 | 7.291 |
| | 4 | ✓ | ✗ | 2.843 | 8.436 | 2.624 | 7.255 |
| | 4 | ✓ | ✓ | **2.841** | **8.432** | **2.622** | **7.234** |

[a] 0.5 represents 50% of the total number of Transformer layers are fine-tuned. Considering some linears are not fine-tuned, the combined buffered activation reduction rate is 2.46×.

[b] 0.2 represents a sparsity of 20% for buffered activation, and since the method uses bitmap storage of sparse values, the overall reduction ratio is 3.81×.

*Table 4.* Algorithm performance on GLUE (Wang et al., 2018) datasets of RoBERTa-Base (Liu et al., 2019) with HyC-LoRA. We report Matthews correlation for COLA and accuracy for the rest (all the larger the better). Due to the unstable convergence of RoBERTa's fine-tuning, we repeated all the experiments 3 times and report the mean and error.

| Quant Bit | Intra | Inter | RTE | COLA | MRPC | SST2 | MNLI | QNLI | QQP | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| 16 | - | - | 77.1±0.2 | 62.8±1.0 | 89.9±0.5 | 94.4±0.4 | 86.6±0.1 | 92.1±0.0 | 90.7±0.1 | 84.8 |
| 4 | ✗ | ✗ | 76.5±1.6 | 61.5±0.6 | 89.4±1.0 | 94.6±0.1 | 84.4±0.5 | 91.8±0.1 | 89.8±0.7 | 84.0 |
| 2 | ✗ | ✗ | 59.4±5.2 | 56.7±0.5 | 81.1±8.8 | 94.5±0.2 | 50.8±22.2 | 50.5±0.0 | 63.2±0.0 | 65.2 |
| 2 | ✓ | ✗ | 75.9±1.0 | 61.1±1.0 | 89.9±2.0 | 94.1±0.2 | 67.8±23.4 | 88.9±3.5 | 90.3±0.1 | 81.1 |
| 2 | ✓ | ✓ | 75.8±0.7 | 61.9±1.5 | 89.6±0.5 | 94.2±0.2 | 85.0±0.1 | 90.7±1.1 | 90.2±0.0 | **83.9** |

Table 2 shows zero-shot accuracy of four different datasets after Math10K fine-tuning. HyC-LoRA@4bit+intra+inter's average accuracy drop (0.10%) is far lower than SparseBP (8.07%), BackRazor (1.90%), and LoRA-FA (4.08%), with similar or even smaller buffered activation consumption.

Table 3 presents the perplexity results for long-sequence tasks. With HyC-LoRA@4-bit+intra+inter setting, perplexity increases are negligible (0.008, 0.052, 0.005, 0.049), demonstrating its efficacy as a memory-efficient approach for fine-tuning on long sequences.

Table 4 presents GLUE benchmark performance. Pure 2-bit quantization causes severe accuracy degradation or even training crashes (e.g., RTE: 77.13 → 59.44; QNLI: 92.05 → 50.54). HyC-LoRA significantly improves accuracy, closely matching the baseline (avg.: Baseline 84.8 vs. HyC-LoRA@2-bit+intra+inter 83.9), which indicates that the fine-tuning of small models such as RoBERTa is particularly sensitive to outlier effects.

We visualize the relationship between the theoretical buffered activation reduction ratio and task accuracy in Figure 6, showing that HyC-LoRA advances the Pareto frontier of memory-efficient fine-tuning.

### 5.3 System Evaluation

#### *5.3.1 Memory Consumption Evaluation*

In Figure 8, we evaluate the actual memory consumption in different fine-tuning methods and training settings mentioned in Section 5.2. Due to the PyTorch (Paszke et al., 2019) framework's coarse-grained memory management, there is a gap between the theoretical analysis results and the measured results. Nevertheless, our method still shows hardware and training platform compatibility. For example, the buffered activation reduction ratio can reach 7.47× when batch=1, seq-length=512 for the Llama-2-7B model. If weight and optimizer are considered simultaneously, the end-to-end memory consumption gain of HyC-LoRA is 1.57×. The end-to-end memory gain from compressing buffered activation will increase with training sequence length and batch size. For example, under batch=4, seq-length=512 and batch=4, seq-length=1024, the end-to-end memory gain can be achieved at 2.87× and 3.97×, respectively. In the extreme case, efficient fine-tuning of the Llama-2-7B model can be accomplished within a memory capacity of 8 GB.

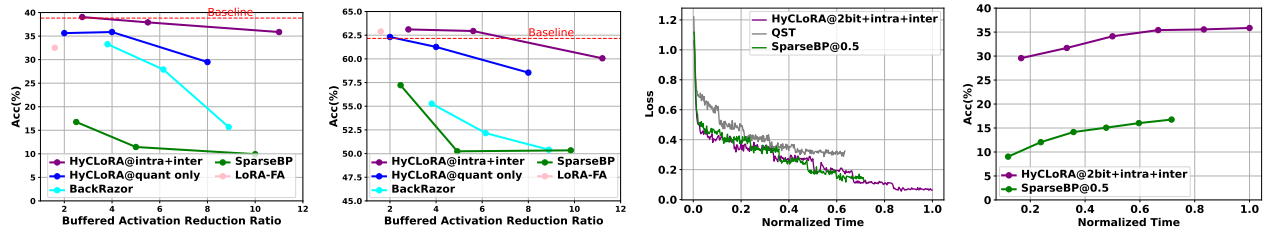
#### *5.3.2 Throughput Evaluation*

In Figure 7, we test the training throughput in different training scenarios on various computing devices, including cloud devices such as NVIDIA A800 80GB PCIe, NVIDIA GeForce RTX 3090, and edge devices like NVIDIA Jetson AGX Orin and NVIDIA Jetson Orin Nano. HyC-LoRA yields a throughput boost of about 1.17× to 1.54× compared to traditional memory-efficient methods like gradient checkpointing (Chen et al., 2016). HyC-LoRA is also faster than compression methods that involve irregular accesses to memory (Jiang et al., 2022).

### 5.4 Discussions

**Iteration Speed vs. Convergence Rate: which matters?** HyC-LoRA does not improve the speed of a single iteration (forward + backward), but can maintain the convergence rate of training compared to other methods. SparseBP (Zhu et al., 2023) blocks the backpropagation computation flow by updating only output-close layers, and QST (Zhang et al., 2024b) by computing backpropagation only for the small side network. Both methods not only reduce buffered activation, but also improve single iteration speed. The effectiveness of the two paradigms has been demonstrated on small networks and simple tasks, but the situation changes in complex generative tasks with large models. Figure 6(c) shows the fine-tuning convergence curves and accuracy gain for three fine-tuning paradigms, we set the horizontal axis to ***normalized time*** instead of ***iteration numbers*** scaling by their throughput, for a comprehensive view of training speed and effect. QST, despite improving the iteration speed, instead requires more computation time when descending the same loss compared to other methods; SparseBP converges faster in training loss than HyC-LoRA, but has a weaker accuracy improvement speed on the validation set than HyC-LoRA as Figure 6(d) shows, which highlights that guaranteeing the integrity of the computational graph in backpropagation is still necessary to achieve high performance in large model's fine-tuning.

**Whether Intra-operator Hybrid Compression is helpful for all operators?** Figure 9(a) shows the relationship between the percentage of outlier channels selected and the quantization error of buffered activation. For buffered activations in the RMSNorm layer, storing only a minimal number of channels in full precision results in a noticeable reduction in quantization error, but this benefit is limited for other operators. We simultaneously perform algorithmic ablation studies on this idea, shown in Table 5(a), further




(a) (b) (c) (d)

*Figure 6.* **(a)(b)**: Visualization of buffered activation reduction ratio vs. GSM8K accuracy (Llama2-7B) and arithmetic tasks average accuracy (Mistral-7B); **(c)**: Training time vs. loss decline on GSM8K dataset; **(d)**: Training time vs. test accuracy on GSM8K dataset.

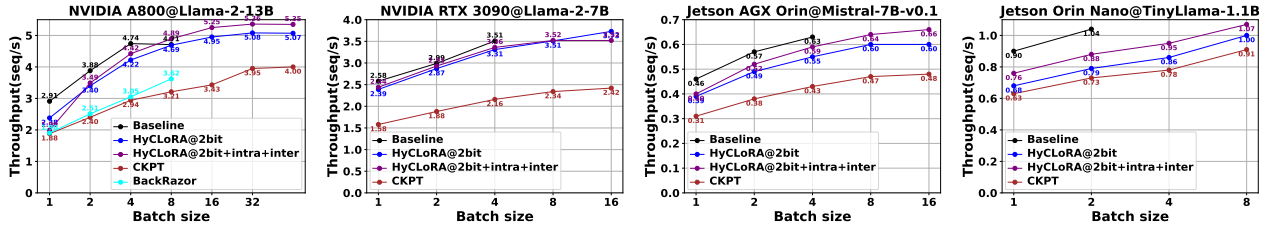
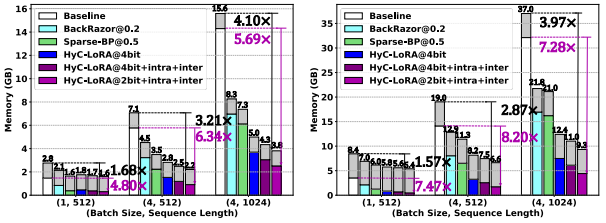
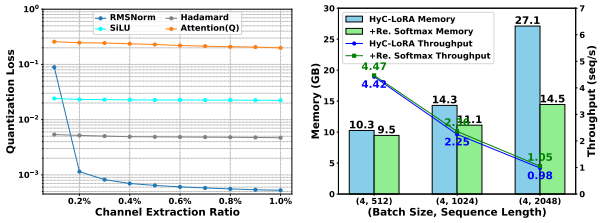



*Figure 7.* Throughput vs. batch size on different devices, the sequence length is 512. CKPT: gradient checkpointing (Chen et al., 2016).




(a) TinyLlama-1.1B (b) Llama-2-7B

*Figure 8. Measured Memory* on NVIDIA A800 platform across different models and training settings. The default weight and optimizer part are marked grey in the chart.




(a) (b)

*Figure 9.* (a): Trend of 2-bit quantization loss and channel extraction ratio across different buffered activations. The buffered activations are sampled from Mistral-7B model in GSM8K task training. (b): End-to-end memory consumption and throughput of the original HyC-LoRA@2bit+intra+inter method with its variant that incorporates recompute softmax (Korthikanti et al., 2023), using the Llama2-13B model on NVIDIA A800 GPU.

demonstrating the necessity of inter-operator hybrid compression.

**Is the calibration stage a burden?** Increasing calibration steps facilitates more robust modeling of compress parameters such as the scaling factor, but leads to a slowdown in training. We show the training accuracy using different calibration data sizes in Table 5(b). The experimental results show that the HyC-LoRA algorithm does not require a large scale of calibration data, and only a few samples can model the buffered activation's distribution.

**Is HyC-LoRA compatible with recompute methods?** Our approach provides a new solution that enables flexible trade-offs between energy consumption, latency, and accuracy in edge fine-tuning scenarios on top of recomputing methods. Furthermore, they can also be stacked together to alleviate peak memory consumption for model fine-tuning. Table 5(c) shows that in the scenario of long sentence fine-tuning, combining standard recomputation (Chen et al., 2016) with our HyC-LoRA method yields an additional 25% reduction in end-to-end memory consumption. In works such as (Korthikanti et al., 2023) use lightweight recomputation to reduce extra overhead – the *sequence parallelism* is designed for large cluster training systems that support distributed algorithm, while its adaptation to single device is not discussed; the *selective activation recomputation* eliminates the need to store the attention map when naive attention is used, which can further reduce memory usage while maintaining throughput and obtain similar accuracy as shown in Figure 9(b) and Table 5(d).

## 6 Related Works

**Model compression for LLMs.** Model compression has been widely used for LLM's deployment. Common model compression methods can be categorized into quantization (Frantar et al., 2022; Xiao et al., 2023; Dettmers et al., 2022; Wei et al., 2022; Lin et al., 2023; Kang et al., 2024) and pruning (Sun et al., 2023; Frantar & Alistarh, 2023; Xia et al., 2023). Some special methods like singular value

*Table 5.* (a): Ablation study of accuracy on intra and inter-operator hybrid compression. intra all.: apply intra-operator hybrid compression to all operators. (b): Calibration dataset size of GSM8K training versus accuracy on Llama-2-7B. The total training steps are 44832, so the overhead of getting statistics in the calibration stage is negligible. (c): Memory consumption (unit: GB) in long sentence fine-tuning decreases further when combining HyC-LoRA@4bit+intra+inter and recompute (Chen et al., 2016). (d): Accuracy on GSM8K when use HyC-LoRA its variant that incorporates recompute softmax.

| bit | 2 | 4 |
|---|---|---|
| ***Llama2-7B*** | | |
| intra all. | 35.10 | 36.47 |
| intra+inter | **35.86** | **37.91** |
| ***Mistral-7B*** | | |
| intra all. | 45.87 | 50.80 |
| intra+inter | **47.16** | **51.63** |

(a)

| Calib. Size | 1 | 5 | 20 |
|---|---|---|---|
| Acc. | 28.96 | **35.86** | 35.71 |

(b)

| seqlen | 8192 | 16384 | 32768 |
|---|---|---|---|
| CKPT | 8.36 | 12.82 | 21.73 |
| +HyC-LoRA | 6.30 | 9.69 | 15.50 |

(c)

| bit | 2 | 4 |
|---|---|---|
| ***Llama2-7B*** | | |
| HyC-LoRA | 35.86 | 37.91 |
| +re. softmax | 34.12 | 38.13 |
| ***Llama2-13B*** | | |
| HyC-LoRA | 44.05 | 47.31 |
| +re. softmax | 42.91 | 47.61 |

(d)

decomposition (SVD) (Wang et al., 2024) and lossless compression (Mao et al., 2024) are also applied in specific scenarios. However, most of the compression works target components in the inference phase, including weights, KV cache, and temporary activations, and less work targets the components of the training process. This work focuses on buffered activation's memory overhead and optimizes LoRA's training memory bottleneck.

**LoRA & its variants.** Low-Rank Adaption (LoRA) (Hu et al., 2021) is currently one of the most popular lightweight fine-tuning methods because of its structural simplicity, no inference overhead, and high interpretability. Based on this, a series of variants of LoRA have been proposed, including different initialization methods (Li et al., 2023; Meng et al., 2024), different learning methods (Hayou et al., 2024; Tian et al., 2024), and changes in the form of trainable parameters (Kopiczko et al., 2023; Liu et al., 2024; Zhao et al., 2024a). However, most existing works focus on enhancing the training effectiveness of LoRA algorithms, or reducing the number of LoRA parameters that are not bottlenecks in memory overheads. In contrast, fewer works analyze the LoRA training process in terms of system overhead. Our work starts from the training system and further reduces the storage of all buffered activations, achieving optimized memory efficiency compared to past works shown in Table 6 **above**.

**Memory efficient training algorithm & system.** Previous works have revealed the curse of massive memory consumption in model training and attempted to compress the optimizer states (Dettmers et al., 2021; Li et al., 2024) or buffered activations (Cai et al., 2020; Chen et al., 2021; Liu et al., 2022; Jiang et al., 2022) in the training stages. Still, few focus on the large language model's fine-tuning scenario. There are also some works on designing more memory-efficient model structures (Sung et al., 2022; Zhang et al., 2024a; Liao et al., 2024) and algorithm-system co-designing (Chen et al., 2016; Ren et al., 2021; Lin et al., 2022; Zhu et al., 2023) to achieve higher memory efficiency. However, these methods are not all designed for edge-side single-device scenarios and may decrease convergence rate and training effectiveness. Our approach performs better than past work on memory optimization for buffered activation, as shown in Table 6 **below**.

*Table 6.* Comparison of training paradigms (above) and buffered activation reduction (below) techniques.

| Method | Weight | Optimizer | Checkpoint | Buffered Activation |
|---|---|---|---|---|
| Full Fine-tuning | Large | Large | Large | Large |
| GaLore (Zhao et al., 2024a) | Large | Small | Large | Large |
| LoRA (Hu et al., 2021) | Large | Small | Small | Large |
| SparseBP (Zhu et al., 2023) | Large | Medium | Medium | Medium |
| QLoRA (Dettmers et al., 2024) | Small | Small | Small | Large |
| **HyC-LoRA** | Small | Small | Small | Small |

| Method | All Operators | Iteration Speed | Convergence Rate | Compress Ratio | No Inference Overhead |
|---|---|---|---|---|---|
| LoRA-FA (Zhang et al., 2023) | No | Medium | Bad | Low | Yes |
| MS-BP (Yang et al., 2024) | No | Medium | Good | Low | Yes |
| BackRazor (Jiang et al., 2022) | Yes | Slow | Bad | High | Yes |
| SparseBP (Zhu et al., 2023) | Yes | Fast | Bad | Low | Yes |
| QST (Zhang et al., 2024b) | Yes | Fast | Bad | High | No |
| CKPT (Chen et al., 2016) | Yes | Slow | Good | High | Yes |
| **HyC-LoRA** | Yes | Medium | Good | High | Yes |

## 7 Conclusion

We present HyC-LoRA, a hybrid compression framework for buffered activations in LoRA training. To reduce the buffered activation's memory consumption, we introduce two hybrid compression mechanisms: intra-operator hybrid compression and inter-operator hybrid compression, and integrate them with a quantization-based buffered activation compression system for further lightweighting of LoRA training. Experiments show that HyC-LoRA can achieve up to 2-bit buffered activation quantization and up to 3.97× end-to-end memory reduction compared to baseline, with negligible accuracy degradation.

## 8 Acknowledgment

This work is supported in part by the National Science and Technology Major Project (2021ZD0114402), and the National Natural Science Foundation of China under Grant 92267203, 62374101.

## REFERENCES


Ansel, J., Yang, E., He, H., Gimelshein, N., Jain, A., Voznesensky, M., Bao, B., Bell, P., Berard, D., Burovski, E., Chauhan, G., Chourdia, A., Constable, W., Desmaison, A., DeVito, Z., Ellison, E., Feng, W., Gong, J., Gschwind, M., Hirsh, B., Huang, S., Kalambarkar, K., Kirsch, L., Lazos, M., Lezcano, M., Liang, Y., Liang, J., Lu, Y., Luk, C. K., Maher, B., Pan, Y., Puhrsch, C., Reso, M., Saroufim, M., Siraichi, M. Y., Suk, H., Zhang, S., Suo, M., Tillet, P., Zhao, X., Wang, E., Zhou, K., Zou, R., Wang, X., Mathews, A., Wen, W., Chanan, G., Wu, P., and Chintala, S. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, pp. 929–947, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703850. doi: 10.1145/3620665.3640366. URL https://doi.org/10.1145/3620665.3640366.

Azerbayev, Z., Ayers, E., and Piotrowski, B. Proof-pile. https://github.com/zhangir-azerbayev/proof-pile, 2024. Accessed on 1 March 2024.

Cai, H., Gan, C., Zhu, L., and Han, S. Tinytl: Reduce memory, not parameters for efficient on-device learning. *Advances in Neural Information Processing Systems*, 33: 11285–11297, 2020.

Chen, J., Zheng, L., Yao, Z., Wang, D., Stoica, I., Mahoney, M., and Gonzalez, J. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In *International Conference on Machine Learning*, pp. 1803–1813. PMLR, 2021.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Chen, Y., Qian, S., Tang, H., Lai, X., Liu, Z., Han, S., and Jia, J. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*, 2023.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Computer, T. Redpajama: An open source recipe to reproduce llama training dataset, April 2023.

Daniel Han, M. H. and team, U. Unsloth, 2023. URL http://github.com/unslothai/unsloth.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Dettmers, T., Lewis, M., Shleifer, S., and Zettlemoyer, L. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*, 2021.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332, 2022.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Diederik, P. K. Adam: A method for stochastic optimization. *(No Title)*, 2014.

Ding, N., Qin, Y., Yang, G., Wei, F., Yang, Z., Su, Y., Hu, S., Chen, Y., Chan, C.-M., Chen, W., et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*, 2022.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *CoRR*, abs/1702.03118, 2017. URL http://arxiv.org/abs/1702.03118.

Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Hayou, S., Ghosh, N., and Yu, B. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

Hong, K., Dai, G., Xu, J., Mao, Q., Li, X., Liu, J., Chen, K., Dong, Y., and Wang, Y. Flashdecoding++: Faster large language model inference on gpus, 2024. URL https://arxiv.org/abs/2311.01282.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Hu, Z., Wang, L., Lan, Y., Xu, W., Lim, E.-P., Bing, L., Xu, X., Poria, S., and Lee, R. LLM-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5254–5276, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.319. URL https://aclanthology.org/2023.emnlp-main.319.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Jiang, Z., Chen, X., Huang, X., Du, X., Zhou, D., and Wang, Z. Back razor: Memory-efficient transfer learning by self-sparsified backpropagation. *Advances in Neural Information Processing Systems*, 35:29248–29261, 2022.

Kang, H., Zhang, Q., Kundu, S., Jeong, G., Liu, Z., Krishna, T., and Zhao, T. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*, 2024.

Koncel-Kedziorski, R., Roy, S., Amini, A., Kushman, N., and Hajishirzi, H. MAWPS: A math word problem repository. In Knight, K., Nenkova, A., and Rambow, O. (eds.), *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1152–1157, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1136. URL https://aclanthology.org/N16-1136.

Kopiczko, D. J., Blankevoort, T., and Asano, Y. M. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*, 2023.

Korthikanti, V. A., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., and Catanzaro, B. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5, 2023.

Lang, T. Triton: Development repository for the triton language and compiler. https://github.com/triton-lang/triton, 2023. Accessed: [Insert Date].

Li, B., Chen, J., and Zhu, J. Memory efficient optimizers with 4-bit states. *Advances in Neural Information Processing Systems*, 36, 2024.

Li, Y., Yu, Y., Liang, C., He, P., Karampatziakis, N., Chen, W., and Zhao, T. Loftq: Lora-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*, 2023.

Liao, B. and Monz, C. Apiq: Finetuning of 2-bit quantized large language model. *arXiv preprint arXiv:2402.05147*, 2024.

Liao, B., Tan, S., and Monz, C. Make pre-trained model reversible: From parameter to memory efficient fine-tuning. *Advances in Neural Information Processing Systems*, 36, 2024.

Lin, J., Zhu, L., Chen, W.-M., Wang, W.-C., Gan, C., and Han, S. On-device training under 256kb memory. *Advances in Neural Information Processing Systems*, 35: 22941–22954, 2022.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Ling, W., Yogatama, D., Dyer, C., and Blunsom, P. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In Barzilay, R. and Kan, M.-Y. (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 158–167, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1015. URL https://aclanthology.org/P17-1015.

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

Liu, X., Zheng, L., Wang, D., Cen, Y., Chen, W., Han, X., Chen, J., Liu, Z., Tang, J., Gonzalez, J., et al. Gact: Activation compressed training for generic network architectures. In *International Conference on Machine Learning*, pp. 14139–14152. PMLR, 2022.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Mao, Y., Wang, W., Du, H., Guan, N., and Xue, C. J. On the compressibility of quantized large language models. *arXiv preprint arXiv:2403.01384*, 2024.

Meng, F., Wang, Z., and Zhang, M. Pissa: Principal singular values and singular vectors adaptation of large language models. *arXiv preprint arXiv:2404.02948*, 2024.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Nagel, M., Fournarakis, M., Amjad, R. A., Bondarenko, Y., van Baalen, M., and Blankevoort, T. A white paper on neural network quantization, 2021. URL https://arxiv.org/abs/2106.08295.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Patel, A., Bhattamishra, S., and Goyal, N. Are NLP models really able to solve simple math word problems? In Toutanova, K., Rumshisky, A., Zettlemoyer, L., Hakkani-Tur, D., Beltagy, I., Bethard, S., Cotterell, R., Chakraborty, T., and Zhou, Y. (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL https://aclanthology.org/2021.naacl-main.168.

Rae, J. W., Potapenko, A., Jayakumar, S. M., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 551–564, 2021.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Sung, Y.-L., Cho, J., and Bansal, M. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *Advances in Neural Information Processing Systems*, 35: 12991–13005, 2022.

Tian, C., Shi, Z., Guo, Z., Li, L., and Xu, C. Hydralora: An asymmetric lora architecture for efficient fine-tuning. *arXiv preprint arXiv:2404.19245*, 2024.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Wang, X., Zheng, Y., Wan, Z., and Zhang, M. Svd-llm: Truncation-aware singular value decomposition for large language model compression. *arXiv preprint arXiv:2403.07378*, 2024.

Wei, X., Zhang, Y., Zhang, X., Gong, R., Zhang, S., Zhang, Q., Yu, F., and Liu, X. Outlier suppression: Pushing the limit of low-bit transformer language models. *Advances in Neural Information Processing Systems*, 35:17402–17414, 2022.

Xia, M., Gao, T., Zeng, Z., and Chen, D. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Yang, Y., Shi, Y., Wang, C., Zhen, X., Shi, Y., and Xu, J. Reducing fine-tuning memory overhead by approximate and memory-sharing backpropagation. *arXiv preprint arXiv:2406.16282*, 2024.

Yu, Z., Shen, L., Ding, L., Tian, X., Chen, Y., and Tao, D. Sheared backpropagation for fine-tuning foundation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5883–5892, 2024.

Zaken, E. B., Ravfogel, S., and Goldberg, Y. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models, 2022. URL https://arxiv.org/abs/2106.10199.

Zhang, B. and Sennrich, R. Root mean square layer normalization. *CoRR*, abs/1910.07467, 2019. URL http://arxiv.org/abs/1910.07467.

Zhang, L., Zhang, L., Shi, S., Chu, X., and Li, B. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023.

Zhang, Z., Zhao, D., Miao, X., Oliaro, G., Li, Q., Jiang, Y., and Jia, Z. Quantized side tuning: Fast and memory-efficient tuning of quantized large language models. *arXiv preprint arXiv:2401.07159*, 2024a.

Zhang, Z., Zhao, D., Miao, X., Oliaro, G., Li, Q., Jiang, Y., and Jia, Z. Quantized side tuning: Fast and memory-efficient tuning of quantized large language models, 2024b. URL https://arxiv.org/abs/2401.07159.

Zhao, J., Zhang, Z., Chen, B., Wang, Z., Anandkumar, A., and Tian, Y. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024a.

Zhao, Y., Lin, C.-Y., Zhu, K., Ye, Z., Chen, L., Zheng, S., Ceze, L., Krishnamurthy, A., Chen, T., and Kasikci, B. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209, 2024b.

Zhu, L., Hu, L., Lin, J., Chen, W.-M., Wang, W.-C., Gan, C., and Han, S. Pockengine: Sparse and efficient fine-tuning in a pocket. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1381–1394, 2023.

# A Experiment Settings

See Table 7 and Table 8.

*Table 7.* Model Configurations

| Model | Hidden dim. | FFN Hidden dim. | Layers | Heads |
|---|---|---|---|---|
| RoBERTa-Base | 768 | 3072 | 12 | 12 |
| TinyLlama-v1.1 | 2048 | 5632 | 22 | 32 |
| Llama-2-7B | 4096 | 11008 | 32 | 32 |
| Llama-2-13B | 5120 | 13824 | 40 | 40 |
| Mistral-7B | 4096 | 14336 | 32 | 32 |

*Table 8.* Training Configurations

| Task | GLUE | GSM8K | Wikitext-2 | Arithmetic Reasoning | Long Sequence Understanding |
|---|---|---|---|---|---|
| Learning Rate | 3e-4 | 3e-4 | 3e-4 | 3e-4 | 2e-5 |
| Max Seq. Length | 128 | 512 | 1024 | 512 | 8192 |
| Batch Size | 32 | 4 | 4 | 4 | 1 |
| Gradient Accu. Steps | 1 | 4 | 4 | 4 | 8 |
| Epochs | 10 | 6 | 3 | 12 | 3 |

# B Compute flow of RoBERTa-Base

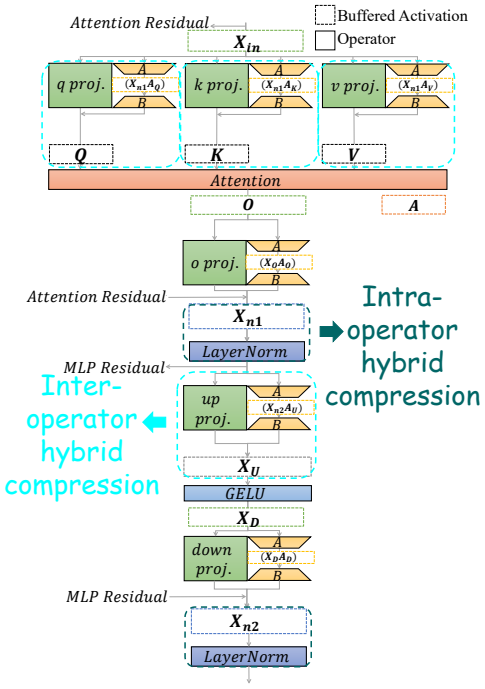
The transformer block architecture in RoBERTa differs slightly from large language models such as Llama series. To illustrate the compatibility of the HyC-LoRA approach, we provide the fine-tuning computational flow of RoBERTa and the modeling of buffered activation in Figure 10. We also follow the approach of inter-operator hybrid compression to perform operator fusion for RoBERTa's FFN layer to reduce the buffered activation, as shown in Figure 11.

# C Visualization of training loss across different HyC-LoRA training settings

We visualize the training loss curve of different experiment settings in Figure 12. During LoRA training using quantized buffered activation, the loss curve shifts significantly upward, causing the model to converge more slowly. With the addition of our proposed method, the loss curve shifts downward, proving its effectiveness in facilitating model learning.

# D Theoretical comparison of different memory-efficient methods

We briefly list and compare the theoretical buffered activation reduction ratio of the methods mentioned in the previous section in Table 9 and Table 10. To simplify the estimation model, we have not considered small pieces of buffered activations that are negligible in the system, such as the scaling factor required for quantization computing, and the parts marked in grey in Figure 3 **Right**.




| Operators | Buffered Activation | Memory Usage |
|---|---|---|
| $Q = X_{n1}(W_Q + A_Q B_Q)$<br>$K = X_{n1}(W_K + A_K B_K)$<br>$V = X_{n1}(W_V + A_V B_V)$ | $X_{n1}$<br>$(X_{n1}A_Q), (X_{n1}A_K)$<br>$(X_{n1}A_V)$ | $(b,s,d)$<br>$3 \times (b,s,r)$ |
| $S = QK^T, A = Softmax(S)$<br>$O = AV$ w/o FlashAttn | $Q,K,V$<br>$A$ | $3 \times (b,s,d)$<br>$(b,h,s,s)$ |
| $O = FlashAttn(Q,K,V)$ w FlashAttn | $Q,K,V$ | $3 \times (b,s,d)$ |
| $X_{n1} = O(W_O + A_O B_O)$ | $O$<br>$(OA_O)$ | $(b,s,d)$<br>$(b,s,r)$ |
| $X_U = LayerNorm(X_{n1})$ | $X_{n1}$<br>$\sigma^2_{n1}, \gamma_{n1}$ | $(b,s,d)$<br>$2 \times (b,s)$ |
| $X_{GELU} = X_U(W_U + A_U B_U)$ | $X_U$<br>$(X_U A_U)$ | $(b,s,d)$<br>$(b,s,r)$ |
| $X_D = GELU(X_G)$ | $X_G$ | $(b,s,d_f)$ |
| $X_{n2} = X_D(W_D + A_D B_D)$ | $X_D$<br>$(X_D A_D)$ | $(b,s,d_f)$<br>$(b,s,r)$ |
| $X_{out} = LayerNorm(X_{n2})$ | $X_{n2}$<br>$\sigma^2_{n2}, \gamma_{n2}$ | $(b,s,d)$<br>$2 \times (b,s)$ |
| ***Estimated Total Size(bit)***<br>*+HyCLoRA@raw quant*<br>*+HyCLoRA@inter + intra* | | $(8d + 2d_f)bsw$<br>$(8d + 2d_f)bsw_q$<br>$(8d + d_f)bsw_q$ |

*Figure 10.* **Left**: Overview computation flow of LoRA training (use RoBERTa transformer block). The figure also specifies the location of buffered activation and indicates the scopes of the two hybrid mechanisms. **Right**: Modeling of buffered activations during training (take RoBERTa-Base as an example). We enumerate *shape* (marked in "()") and *bit-width* (marked in "[]") of related operators. Operators that do not need buffered activations like residual connection and reshaping are not listed for simplicity. Some small chunks of buffered activation have a negligible share of memory, so we do not compress them and they are marked gray in the table. $d$: hidden dimension; $d_f$: FFN's hidden dimension; $s$: training sequence length; $h$: head dimension; $w$: original bit-width of buffered activation; $w_q$: quantization bit-width.

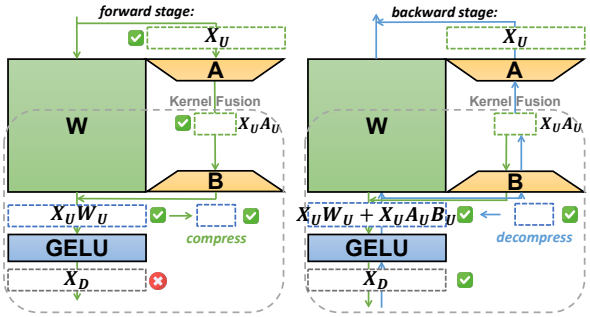



*Figure 11.* Operator fusion of FFN layer in RoBERTa.

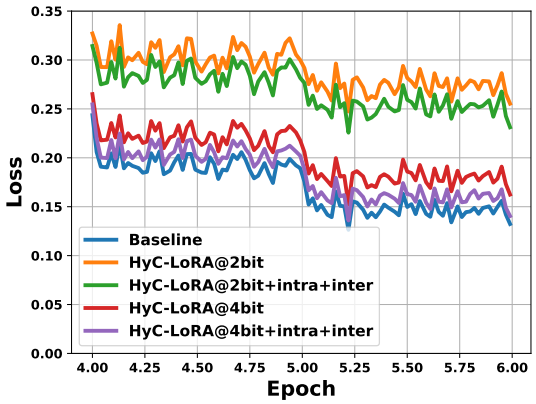



*Figure 12.* Loss convergence speed under different settings of TinyLlama-v1.1's fine-tuning on GSM8K.

*Table 9.* Theoretical buffered activation memory consumption of different training methods. $d$: hidden dimension; $d_f$: FFN's hidden dimension; $s$: training sequence length; $h$: head dimension; $w$: original bit-width of buffered activation; $w_q$: quantization bit-width; $x\%$: Fine-tuned transformer layer ratio for SparseBP method and sparsity ratio for BackRazor method. We freeze the o proj, down proj for SparseBP method. We save a 1-bit bitmap additionally for BackRazor method.

| **Method** | RoBERTa arch. | Llama-2 arch. |
|---|---|---|
| Baseline(QLoRA) (Dettmers et al., 2024) | $(8d + 2d_f)bsw$ | $(8d + 4d_f)bsw$ |
| LoRA-FA (Zhang et al., 2023) | $(5d + d_f)bsw$ | $(5d + 3d_f)bsw$ |
| SparseBP@$x\%$ (Zhu et al., 2023) | $(7d + d_f)bsw \times x\%$ | $(7d + 3d_f)bsw \times x\%$ |
| BackRazor@$x\%$ (Jiang et al., 2022) | $(8d + 2d_f)bs(w \times x\% + 1)$ | $(8d + 4d_f)bs(w \times x\% + 1)$ |
| HyC-LoRA@$w_q$bit | $(8d + 2d_f)bsw_q$ | $(8d + 4d_f)bsw$ |
| HyC-LoRA@$w_q$bit+intra+inter | $(8d + d_f)bsw_q$ | $(8d + 2d_f)bsw$ |

*Table 10.* Theoretical buffered activation memory reduction ratio of different models and training configurations. The compressed ratio is calculated using the model parameters in Table 7 and the formula in Table 9.

| **Method** | RoBERTa-Base | Llama-2-7B |
|---|---|---|
| Baseline(QLoRA) (Dettmers et al., 2024) | 1.00× | 1.00× |
| LoRA-FA (Zhang et al., 2023) | 1.78× | 1.43× |
| SparseBP@50% (Zhu et al., 2023) | 2.91× | 2.49× |
| SparseBP@25% (Zhu et al., 2023) | 5.82× | 4.98× |
| BackRazor@20% (Jiang et al., 2022) | 3.81× | 3.81× |
| BackRazor@10% (Jiang et al., 2022) | 6.15× | 6.15× |
| HyC-LoRA@4-bit | 4.00× | 4.00× |
| HyC-LoRA@4-bit+intra+inter | 5.33× | 5.61× |
| HyC-LoRA@2-bit | 8.00× | 8.00× |
| HyC-LoRA@2-bit+intra+inter | 10.67× | 11.21× |

# A. Artifact Appendix

## A.1 Abstract

*The artifact appendix provides the dataset, code, procedure, hyper-parameter settings, etc. needed to reproduce the HyC-LoRA project. The overall training framework is implemented by PyTorch, some of the operators are implemented by OpenAI Triton, and can be run on the Linux operating system with NVIDIA GPU support. This artifact appendix evaluates the HyC-LoRA project's code availability, algorithmic effectiveness, and system performance. Github project: `https://github.com/Ther-nullptr/HyC-LoRA-release`*

## A.2 Artifact check-list (meta-information)

- **Program:** Python 3.10+
- **Models:** Checkpoints from huggingface: llama-2-7b (`https://huggingface.co/meta-llama/Llama-2-7b-hf`), llama-2-13b (`https://huggingface.co/meta-llama/Llama-2-13b-hf`), mistral-7b (`https://huggingface.co/mistralai/Mistral-7B-v0.1`), roberta-base (`https://huggingface.co/FacebookAI/roberta-base`)
- **Dataset:** *Auto download from huggingface*: GSM8K, Wikitext-2, GLUE; *Download use script*: Math10K, SVAMP, mawps, AQuA; *Download from google drive*: redpajama, proof-pile, PG-19. The download path can be seen in Github project.
- **Run-time environment:** Ubuntu 20.04 LTS, CUDA Version 12.0+, PyTorch version 2.0+
- **Hardware:** CPU: x86 architecture; GPU: NVIDIA GPU (Ampere arch. is recommend) with 16GB+ memory.
- **Execution:** See the `README.md` in attachments provided
- **Metrics:** loss, accuracy, memory consumption, throughput
- **Output:** training log, task accuracy/perplexity, checkpoints
- **Experiments:** See **A.5 Experiment workflow, Evaluation and expected result** part.
- **How much disk space required (approximately)?:** 200GB+
- **How much time is needed to prepare workflow (approximately)?:** Download the checkpoints/datasets: 30min; Build the Python environment: 10min.
- **How much time is needed to complete experiments (approximately)?:** Full experiment: about 2-3 days. Demo experiment: about 2h.
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** Apache License 2.0
- **Data licenses (if publicly available)?:** MIT license
- **Workflow framework used?:** No

## A.3 Description

### A.3.1 How delivered

The artifact is distributed as an environment configuration manual. A companion GitHub repository `https://github.com/Ther-nullptr/HyC-LoRA-release` contains:

- Source code
- Detailed descriptions of scripts
- Links of models and datasets

### A.3.2 Hardware dependencies

- CPU: x86 architecture. Since the computation in this project mainly relies on the GPU, there is no special requirement for CPU performance. It is worth noting that although some of our experiments were conducted on devices with **arm** architecture (e.g., NVIDIA Jetson Orin series), the open-source repository code is mainly compatible with **x86** architecture.
- GPU: Proved GPUs: **NVIDIA A800 80GB PCIe, NVIDIA RTX 6000 Ada, NVIDIA GeForce RTX 3090**. Other GPUs were not tested, but might work if meeting the following conditions: ❶ HBM: 16GB+ is recommended, more specific minimum memory requirements can be obtained from **Figure 8** in the original paper. ❷: Arch: Ampere or Hopper architecture for bf16/tf32 tensor core support.
- Disk: 200GB+: models (about 70GB), datasets (about 50GB) and rest for generated checkpoints.

### A.3.3 Software dependencies

See **Quick Start** part in `README.md` of Github Project.

### A.3.4 Datasets

- GSM8K: experiments for **Table 3**
- Wikitext2: experiments for **Table 3**
- Math10K, SVAMP, mawps, AQuA: experiments for **Table 4**
- redpajama, proof-pile, PG-19: experiments for **Table 5**
- GLUE: experiments for **Table 6**

## A.4 Installation

See **Quick Start** part in `README.md` of Github Project. One possible organization of the directory tree is as follows:

**Listing 1.** dir tree

```
.
|-- figures # Thesis illustration
|   |-- main-intra-inter.jpg
|   |-- main-intra-inter.pdf
|-- models # Model structure implementation
|   |-- llama
|   |-- llama_flash_attn
|   |-- mistral
|   |-- roberta
|   |-- utils
|       |-- compute_utils.py
|-- operators # Triton kernels for certain operator
|   |-- compress_function_kernel.py
..   ..
|   |--
    triton_fuse_lora_silu_hadamard_forward_kernels.py
|-- ckpt # Needs to be created on your own, for
    placing models checkpoints (e.g. llama-2-7b)
|-- README.md
|-- requirements.txt # Dependency libraries required
    by the project
|-- download_dataset.sh # Download the datasets for `
    run_multitask.sh`
|-- run_glue.py
|-- run_glue.sh
|-- run_gsm8k.py # main process of gsm8k experiment
|-- run_gsm8k.sh # script for running gsm8k
    experiment
|-- run_longseq.py
|-- run_longseq.sh
|-- run_multitask.py
|-- run_multitask.sh
|-- run_wikitext2.py
|-- run_wikitext2.sh
```

```
|-- utils # utils functions
    |-- accuracy
    |-- glue
    |-- longseq
    |-- math_10k
```

### A.5 Experiment workflow, Evaluation and expected result

*Full experiments* require a lot of arithmetic and time. If the device has less arithmetic or limited time, try *demo experiments*.

**(E1) Basic Algorithm Experiments – demo** *[5 min human-time and about 10 min compute-time]*

- ***Preparation:*** ❶ Setup the environment following **Quick Start** part in `README.md`. ❷ Download the models on huggingface, e.g., llama-2-7b (`https://huggingface.co/meta-llama/Llama-2-7b-hf`), and put the model dir to a certain position e.g., `ckpt/`. ❸ Change the `model_name` and `model_dir` config in `run_gsm8k.sh` to your own path like:

  ```
  model_name=<your model name>
  model_dir=<your model dir>
  model_name_full=${model_dir}/${model_name}
  ```

- ***Execution:*** Run `bash run_gsm8k.sh`
- ***Results:***
  ❶ After launch, **the configuration of the core parameters** will be printed on the terminal in green color:

  ```
  ********** HycLora Configuration **********
  [INFO] HycLora type: intra_inter
  [INFO] Iteration Threshold: 5
  [INFO] Softmax Outlier Ratio: 0.05
  [INFO] Layernorm Outlier Ratio: 0.005
  [INFO] Quantization Bit: 2
  ******************************************
  ```

  ❷ After a few minutes, a **progress bar** recording the training process and **degrading model loss record** will appear on the terminal, proving that the model is training properly (then the program can be canceled manually):

  ```
  {'loss': 1.0592, 'grad_norm':
      0.158203125, 'learning_rate':
      3.529411764705882e-05, 'epoch':
      0.02}
  {'loss': 0.9622, 'grad_norm': 0.234375,
      'learning_rate': 7.058823529411764e
      -05, 'epoch': 0.04}
  {'loss': 0.682, 'grad_norm': 0.375, '
      learning_rate':
      0.00010588235294117647, 'epoch':
      0.06}
  {'loss': 0.5484, 'grad_norm':
      0.1689453125, 'learning_rate':
      0.00014117647058823528, 'epoch':
      0.09}
    2%|--                                |
        49/2802 [02:48<1:19:14,  1.73s/it]
  ```

- ***Notes:*** The first time you start the programme, it will automatically download the required datasets, which may take a few minutes, so please ensure you have a good internet connection. The program may display the following error if your network encounters an issue. In this case, please check your network connection:

  ```
  Traceback (most recent call last):
    File "/home/abc/HyC-LoRA-release/run_gsm8k.py",
        line 569, in <module>
      train_and_eval()
    ...
    File "/home/abc/anaconda3/envs/hyclora/lib/
        python3.10/site-packages/datasets/load.py",
        line 1780, in dataset_module_factory
      raise ConnectionError(f"Couldn't reach '{path
          }' on the Hub ({type(e).__name__})")
  ConnectionError: Couldn't reach 'gsm8k' on the Hub
      (SSLError)
  ```

**(E2) Basic Algorithm Experiments – full** *[5 min human-time and about 1.5 h compute-time on A800 / about 3 h compute-time on 3090]*

- ***Preparation:*** Same as **E1**.
- ***Execution:*** Same as **E1**.
- ***Results:***
  ❶ When the model has been trained, the terminal will display a summary of the training phase:

  ```
  {'loss': 0.145, 'grad_norm': 0.296875, '
      learning_rate': 4.8529260605706386e
      -08, 'epoch': 5.95}
  {'loss': 0.1338, 'grad_norm':
      0.27734375, 'learning_rate':
      1.4439004654120956e-08, 'epoch':
      5.97}
  {'loss': 0.1266, 'grad_norm':
      0.30859375, 'learning_rate':
      4.0108971875452144e-10, 'epoch':
      5.99}
  {'train_runtime': 4772.7021, '
      train_samples_per_second': 9.395, '
      train_steps_per_second': 0.587, '
      train_loss': 0.28048270989706653, '
      epoch': 6.0}
  ```

  ❷ After that, the script will step into evaluation stage. Final evaluation result will be displayed on the terminal:

  ```
  prediction [18.0, 3.0, ..., 11.0]
  ground truth [18.0, 3.0, ..., 14.0]
  adapter: None | GSM8K test accuracy:
      29.42% | full precision: False
  ```

  ❸ A `.log` file will be generated in `exp_results_gsm8k/`, recording the configuration of the experiment, the training process and the evaluation process.
- ***Corresponding Content:*** **Table 3: Algorithm performance ... (the same as below)** in original paper.
- ***Notes:*** If you want to complete the flow of the full experiment while saving time, you can modify the `num_train_epochs` config in `run_gsm8k.sh`.

**(E3) Reset Configurations – demo/full** *[time consumptions same as **E1**]*

- ***Preparation:*** Same as **E1**.
- ***Execution:***
  ❶ Modify the core HyC-LoRA hyper-parameters in `run_gsm8k.sh` script:

```
use_hyclora=True
layer_type=intra_inter
iteration_threshold=5
softmax_outlier_ratio=0.05
layernorm_outlier_ratio=0.005
q_bit=2
```

The exact meaning of the hyper-parameters can be found in the **Configuration Guide** part in `README.md`.

❷ Rerun `bash run_gsm8k.sh`

- ***Result:***
  Same as **E1** and **E2**. The training and evaluation logs change with configuration changes.
- ***Corresponding Content:*** **Table 3: Algorithm performance ... (the same as below)** in original paper.

**(E4) Memory Consumption Evaluation** *[5 min human-time and about 5 min compute-time]*

- ***Preparation:*** Same as **E1** (for `run_memory_throughput.sh`).
- ***Execution:***
  ❶ Modify the evaluation config and the core HyC-LoRA hyper-parameters in `run_memory_throughput.sh` script:

```
#! evaluation config
# evaluation sequence length
seq_len=512
# evaluation batch size
per_device_train_batch_size=4
# whether to evaluate memory or
    throughput
evaluate_memory=True
evaluate_throughput=False

#! HyCLoRA core parameters
use_hyclora=True
layer_type=intra_inter
iteration_threshold=5
softmax_outlier_ratio=0.05
layernorm_outlier_ratio=0.005
q_bit=2
```

❷ Run `bash run_memory_throughput.sh`

- ***Result:***
  The profiled memory consumption should be printed in the terminal:

```
  0%|                    | 0/2802 [00:00<?,
      ?it/s]
torch.cuda.memory_allocated (static):
    4327.07 MiB
torch.cuda.memory_allocated: 6045.48 MiB
torch.cuda.memory_allocated: 6134.20 MiB
```

  The program will automatically stop after two iterations.
- ***Corresponding Content:*** **Figure 8: Measured Memory...** in original paper.
- ***Notes:*** The "(static)" part means the **static memory allocation** during training, including the weight and optimizer state; The subsequent data represent the **dynamic peak memory allocation** after considering the buffered activation memory consumption. The two above can be subtracted to get the buffered activation memory usage.

**(E5) Throughput Evaluation** *[5 min human-time and about 10 min compute-time]*

- ***Preparation:*** Same as **E1**.
- ***Execution:*** Modify the evaluation config and the core HyC-LoRA hyper-parameters in `run_memory_throughput.sh` script:

```
# whether to evaluate memory or
    throughput
evaluate_memory=False
evaluate_throughput=True
```

- ***Result:*** The training throughput data can be read from the right side of the progress bar:

```
1%|-   |20/2802[02:06<1:56:26,2.51s/it]
1%|-   |30/2802[02:31<1:55:34,2.50s/it]
1%|--  |40/2802[02:56<1:54:51,2.50s/it]
2%|--  |49/2802[03:18<1:54:23,2.49s/it]
```

  In the original paper's setting, per iteration has 16 sequences (`per_device_train_batch_size=4` and `gradient_accu_steps=4`), so the sequence per second can be calculated as: $\frac{16}{\text{second per iteration}}$. After the data has been read, the program can be canceled manually.
- ***Corresponding Content:*** **Figure 7: Throughput...** in original paper.
- ***Notes:*** Due to the **cold start** and **short calibration phase** of the machine, we strongly recommend reading the throughput data after the operation has stabilized (after about 20-30 iterations).

## A.6 Experiment customization

None.

## A.7 Notes

None.

## A.8 Methodology

Submission, reviewing and badging methodology:

- `http://cTuning.org/ae/submission-20190109.html`
- `http://cTuning.org/ae/reviewing-20190109.html`
- `https://www.acm.org/publications/policies/artifact-review-ba`