BLOC: BLOCKWISE COMPOSITION FOR ADAPTIVE-RANK FINE-TUNING IN LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Parameter-efficient fine-tuning (PEFT) critically depends on how weight updates are parameterized with a small number of trainable parameters. A common PEFT approach assumes that the weight update are formulated as a transformation of the product of two or more low-dimensional matrices. Such parameterizations often impose structural assumptions on the resulting update, such as favoring low- or high-rank solutions, rather than allowing the realized rank to emerge from optimization and vary adaptively across layers. To address this limitation, we propose **BLOC**, a PEFT method based on *BLOckwise Composition*. Specifically, BLOC partitions each weight update matrix into smaller blocks and constructs each block as a sparse linear combination of trainable block components defined directly in the block space. To control the number of trainable parameters, BLOC shares these components across compatible Transformer layers and uses a one-time gradient-guided calibration for task-informed initialization. Across three backbones and diverse reasoning tasks, BLOC achieves strong performance, while producing adaptive-rank updates and effectively reusing shared block components across layers.

1 INTRODUCTION

Parameter-efficient fine-tuning (PEFT) techniques adapt large language models (LLMs) to downstream tasks by using only a small number of trainable parameters while keeping the pretrained LLMs frozen. LoRA (Hu et al., 2022) and its variants (Zhang et al., 2023; Liu et al., 2024a; Hayou et al., 2024) parameterize weight updates using the product of low-rank matrices, where their ranks determine the number of trainable parameters and constrain the resulting weight update. More recent PEFT approaches (Huang et al., 2025; Jiang et al., 2024; Albert et al., 2025) further modify the parameterization of the weight update to support more expressive and potentially higher-rank weight updates while retaining parameter efficiency.

Currently, most PEFT methods achieve the parameter efficiency through matrix factorization or some variants. Specifically, in those methods, the weight update is typically formulated as a transformation (e.g., a scalar transformation in LoRA or Hadamard product in HiRA (Huang et al., 2025)) of the product of two or more low-rank trainable matrices. Although effective, such formulation on the weight update implicitly poses some assumption on its structure. For example, LoRA constrains the update to be low-rank, whereas HiRA is designed to produce high-rank updates, as reflected in the layer-wise numerical ranks shown in Figure 1. Since a fixed or uniform rank across layers may be suboptimal (Zhang et al., 2024; Liu et al., 2024b), this motivates a more flexible parameterization in which the realized update rank is not explicitly prescribed, but instead emerges from the learned composition, enabling adaptive layer-wise rank variation.

To achieve that, we propose **BLOC**, a BLOckwise Composition for the weight update in PEFT. BLOC partitions the weight update into smaller update blocks and introduces trainable components directly in the corresponding block space. Rather than using matrix multiplication of low-rank matrices, each block in BLOC is formed as a sparse linear combination of different block components. Note that we do not pose any constraint or assumption on the rank of each component, making it possible to achieve adaptive rank. As illustrated in Figure 2, different component selections and coefficients allow the shared components to construct distinct block updates, which are then assembled into the full weight update.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

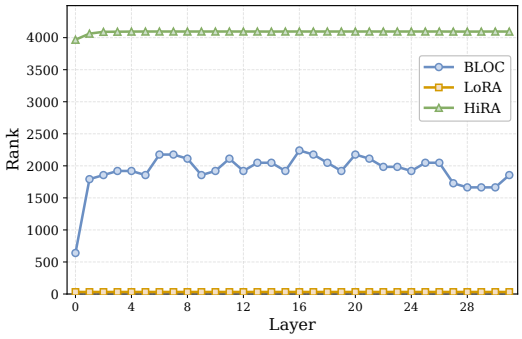


Figure 1: **Layer-wise rank of W_q updates.** Under a matched parameter budget, LoRA remains constrained to its fixed low-rank structure, HiRA produces nearly full ranks across layers, while BLOC exhibits heterogeneous intermediate ranks, suggesting more adaptive use of update capacity across layers.

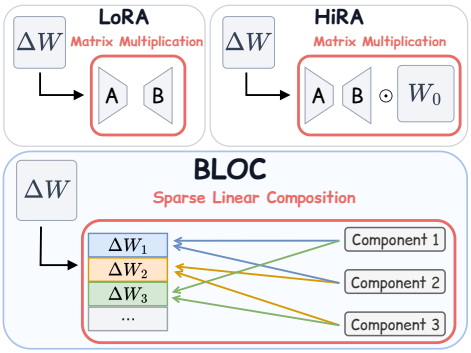


Figure 2: **Parameterization of BLOC.** BLOC constructs block updates through sparse linear composition of trainable block components defined directly in the block update space.

Although BLOC uses a small number of block components, maintaining an independent component bank for every adapted layer would still cause the number of trainable parameters to scale with the number of layers. To resolve this issue, BLOC shares a component bank across Transformer layers, while retaining separate block components selections for individual blocks. This enables the same components to support distinct block updates across layers. In the training process of BLOC, we perform a one-time gradient-guided calibration to initialize the shared block components, selections and coefficients from task-specific gradients, after which the components and local coefficients are jointly optimized. Across three backbones and commonsense, mathematical, and code reasoning tasks, BLOC achieves superior performance over representative baseline methods under comparable trainable-parameter budgets. Further analyses show that BLOC produces adaptive-rank updates, effectively utilizes additional component capacity as the parameter budget increases, and reuses shared components across layers.

The contributions of this work are three-fold.

- We propose **BLOC**, a blockwise compositional parameterization for PEFT that constructs weight updates through linear combinations of trainable block components, providing an alternative to multiplication-based parameterization.
- BLOC further incorporates cross-layer component sharing to control the trainable-parameter cost, together with a one-time gradient-guided calibration that provides a task-informed initialization before fine-tuning.
- Across three backbones and diverse reasoning tasks, BLOC consistently performs strongly under matched parameter budgets. Further analyses reveal broad spectral dimensionality and extensive cross-layer component reuse.

2 Related Work

Weight-Update Parameterization. LoRA (Hu et al., 2022) parameterizes weight updates with low-rank matrix factors, while variants such as DoRA (Liu et al., 2024a) further modify this factor-based formulation. Recent methods explore more expressive or higher-rank update constructions. HiRA (Huang et al., 2025) introduces Hadamard interactions between low-rank factors, MoRA (Jiang et al., 2024) maps a trainable square matrix to the update through fixed transformations, and RandLoRA (Albert et al., 2025) combines random low-rank bases to construct full-rank updates. Despite their different designs, these methods generally construct weight updates through factor-based representations or transformations from auxiliary parameter spaces. BLOC instead uses blockwise linear composition, constructing each block directly from trainable matrix components defined in the same update space.

Cross-Layer Parameter Reuse. Parameter sharing further improves parameter efficiency by reusing adaptation parameters across layers or modules. VeRA (Kopiczko et al., 2024) shares frozen low-rank projection matrices across layers and learns lightweight scaling vectors. VB-LoRA (Li et al., 2024) constructs LoRA factors from a shared vector bank, while LoTR (Bershatsky et al., 2024) shares tensor-factorized adaptation components across layers. While these methods demonstrate the effectiveness of cross-layer parameter reuse, their shared adaptation structures remain based on low-rank or factorized parameterizations. BLOC instead shares trainable block components across compatible Transformer layers, enabling parameter reuse while supporting substantially adaptive-rank weight updates.

Task-Informed Initialization. Initialization can improve PEFT optimization by providing a task-relevant starting point. PiSSA (Meng et al., 2025) initializes low-rank adapters from the principal singular components of pretrained weights, while LoRA-GA (Wang et al., 2024) uses downstream task gradients to initialize LoRA factors. BLOC similarly uses downstream gradients, but applies them to its blockwise compositional parameterization. A one-time calibration initializes the shared matrix components and coefficients, determines the block component selections before fine-tuning.

3 Methodology

3.1 Problem Setup and Design Overview

Consider a set of pretrained linear projections with the same weight shape across L Transformer layers, where $\ell \in \{1, \dots, L\}$ denotes the layer index. For a frozen pretrained weight $\mathbf{W}_{0,\ell} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ in layer ℓ, fine-tuning learns a weight update $\Delta\mathbf{W}_\ell \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. Given an input $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$, the adapted projection produces an output $\mathbf{y} \in \mathbb{R}^{d_{\text{out}}}$ as

$$\mathbf{y} = (\mathbf{W}_{0,\ell} + \Delta\mathbf{W}_\ell)\,\mathbf{x}. \tag{1}$$

PEFT reduces the number of trainable parameters by introducing a structured parameterization of $\Delta\mathbf{W}_\ell$. For example, LoRA constructs the update from lower-dimensional matrix factors through matrix multiplication. Instead, the proposed BLOC method uses a blockwise compositional parameterization, as illustrated in Figure 3. It partitions $\Delta\mathbf{W}_\ell$ into smaller blocks and constructs each block as a sparse linear combination of trainable block components. These components are shared across Transformer layers, while each block retains its own selection and coefficients.

3.2 Sparse Blockwise Composition from Reusable Block Components

We consider the attention projections W_q, W_k, and W_v, and the feed-forward projections W_{up} and W_{down}. Let $p \in \{q, k, v, \text{up}, \text{down}\}$ denote the projection type, and let $\Delta\mathbf{W}_\ell^p \in \mathbb{R}^{d_{\text{out}}^p \times d_{\text{in}}^p}$ denote the corresponding weight update at layer ℓ. BLOC partitions the update along the output dimension for $p \in \{q, k, v, \text{up}\}$ and along the input dimension for $p = \text{down}$. This preserves the full Transformer hidden dimension within each local block. Since the same parameterization is applied independently to all projection types, we omit the projection index p for simplicity and use d to denote the dimension that is not partitioned. Thus, $d = d_{\text{in}}^p$ for $p \in \{q, k, v, \text{up}\}$ and $d = d_{\text{out}}^p$ for $p = \text{down}$.

Under this partitioning, each update is represented by N_ℓ local blocks $\Delta\mathbf{W}_{\ell,i} \in \mathbb{R}^{b \times d}$, where $i \in \{1, \dots, N_\ell\}$ indexes the blocks and b denotes the block size along the partitioned dimension. For W_{down}, this common $b \times d$ notation is applied to the transposed update for convenience. The local blocks are concatenated along the partitioned dimension to recover the corresponding $\Delta\mathbf{W}_\ell$.

Rather than learning each $\Delta\mathbf{W}_{\ell,i}$ independently, BLOC constructs it as a sparse linear combination of trainable block components. For each projection type, these block components form a components bank $\mathcal{B} = \{\mathbf{B}_j\}_{j=1}^{S}$ shared across layers, where $\mathbf{B}_j \in \mathbb{R}^{b \times d}$ denotes the j-th block component and S is the size of the bank. Each block selects K components from the shared bank, with the selected indices denoted by $\mathcal{S}_{\ell,i}$. The selection is determined during task-informed initialization, as described in Section 3.4, and remains fixed throughout fine-tuning. The i-th update block at layer ℓ is then constructed as a weighted sum of the selected components:

$$\Delta\mathbf{W}_{\ell,i} = \sum_{j \in \mathcal{S}_{\ell,i}} c_{\ell,i,j}\mathbf{B}_j, \tag{2}$$

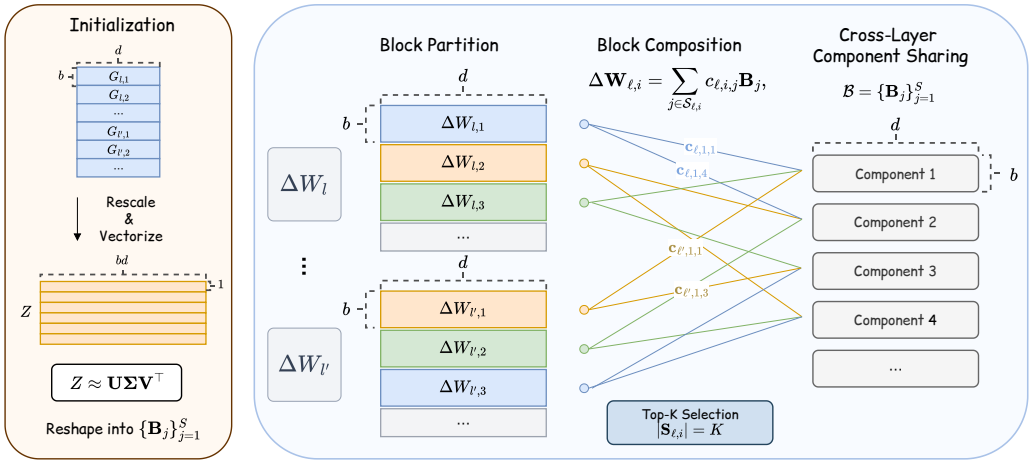


Figure 3: **Overview of BLOC.** Left: calibration gradients from layers are rescaled, vectorized, and stacked, and SVD is used to initialize the shared matrix components. Right: each $\Delta\mathbf{W}$ is partitioned into blocks, and each block is constructed as a Top-K sparse linear combination of same-shaped block components from a bank shared across layers.

where $|\mathcal{S}_{\ell,i}| = K$ and $c_{\ell,i,j}$ denotes the trainable coefficient associated with component $\mathbf{B}_j$. Different selections and coefficients allow the shared block components to form distinct block updates.

The resulting blocks are concatenated along the partitioned dimension to recover the full update $\Delta\mathbf{W}_\ell$. Because a component can be selected by multiple blocks across layers, its parameters are reused across different local updates, improving parameter efficiency.

3.3 Rank Capacity and Parameter Cost

Although BLOC does not explicitly prescribe the realized update rank through its parameterization, the finite component bank still constrains the rank capacity of the resulting update. To characterize this capacity, we vertically stack the S block components as $\mathbf{B}_{\text{stack}} = [\mathbf{B}_1^\top, \ldots, \mathbf{B}_S^\top]^\top \in \mathbb{R}^{Sb\times d}$. The full update $\Delta\mathbf{W}_\ell$ consists of N_ℓ blocks of size $b \times d$, and thus has $N_\ell b$ rows. Since each block is a linear combination of the shared components, every row of $\Delta\mathbf{W}_\ell$ lies in the row space of $\mathbf{B}_{\text{stack}}$. Therefore, its rank is bounded by both the number of rows and the dimensionality of the row space spanned by the shared block component bank:

$$\operatorname{rank}(\Delta\mathbf{W}_\ell) \le \min\left\{N_\ell b,\ \operatorname{rank}(\mathbf{B}_{\text{stack}})\right\} \le \min\left\{N_\ell b,\ Sb,\ d\right\}. \tag{3}$$

The same bound applies to W_{down} because transposition preserves matrix rank. Unlike factorized adapters in which a factor dimension directly specifies a rank constraint, BLOC has its rank capacity from the shared block component bank and block size. The rank realized after fine-tuning is therefore determined by the learned components and their compositions within this capacity.

Cross-layer sharing keeps BLOC parameter efficient. For each projection type, the number of trainable parameters is

$$P_{\text{BLOC}} = Sbd + K\sum_{\ell=1}^{L} N_\ell. \tag{4}$$

The Sbd parameters of the shared component bank are stored once and reused across all L layers, while the $\sum_{\ell=1}^{L} N_\ell$ blocks each introduce only K trainable coefficients. The total number of trainable parameters in BLOC is obtained by summing Eq. (4) over all projection types.

3.4 Task-Informed Initialization

Before fine-tuning, BLOC performs a one-time gradient calibration independently for each projection type to initialize the shared block components and coefficients. The pretrained weights remain unchanged during calibration.

Algorithm 1 Algorithm for BLOC

Require: Frozen pretrained weights $\mathbf{W}_{0,\ell}$; training data $\mathcal{D}_{\text{train}}$; block size b, number of components S, number of selected components K, scaling exponent β, calibration batches M

Collect calibration gradients from the first M minibatches (Eq. 5)
for each projection type **do**
  Partition $\mathbf{G}_\ell$ using the same block layout as $\Delta\mathbf{W}_\ell$ and rescale the blocks (Eq. 6)
  Stack the vectorized rescaled blocks from all layers into $\mathbf{Z}$
  Compute truncated SVD and initialize the shared matrix components:

$$\mathbf{Z} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top, \qquad \mathbf{B}_j \leftarrow \text{reshape}_{b\times d}(\mathbf{v}_j)$$

  For each block, select K components
  Initialize the corresponding coefficients $c_{\ell,i,j}$ using the normalized signed cosine scores
end for
Fine-tune the shared block components and selected coefficients with $\mathbf{W}_{0,\ell}$ frozen
return trained BLOC parameters

Gradient calibration. We compute gradients with respect to each target weight $\mathbf{W}_{0,\ell}$ and average them over the first M minibatches of the downstream training set as

$$\mathbf{G}_\ell = \frac{1}{M}\sum_{m=1}^{M} \frac{\partial \mathcal{L}_m}{\partial \mathbf{W}_{0,\ell}}, \tag{5}$$

where $\mathcal{L}_m$ denotes the language-modeling loss on minibatch m. No optimizer step is taken during the calibration. We partition $\mathbf{G}_\ell$ in the same way as $\Delta\mathbf{W}_\ell$, yielding gradient blocks $\mathbf{G}_{\ell,i} \in \mathbb{R}^{b\times d}$. To control the influence of gradient magnitude across blocks while preserving their task-specific directions, each gradient block is rescaled as

$$\widetilde{\mathbf{G}}_{\ell,i} = \frac{-\mathbf{G}_{\ell,i}}{\max(\|\mathbf{G}_{\ell,i}\|_F, \epsilon)^\beta}, \tag{6}$$

where the negative sign gives the descent direction and ϵ ensures numerical stability. The exponent $\beta \in [0,1]$ controls the degree of magnitude normalization. When $\beta = 0$, the original gradient magnitude is preserved, while $\beta = 1$ normalizes each block to retain its direction only.

Shared-bank initialization. For each projection type, we initialize the shared component bank from gradient information across Transformer layers. Specifically, the rescaled gradient blocks are vectorized and stacked row-wise into $\mathbf{Z} \in \mathbb{R}^{(\sum_{\ell=1}^{L} N_\ell)\times bd}$, where each row represents one gradient block. We then apply singular value decomposition to $\mathbf{Z}$:

$$\mathbf{Z} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top, \tag{7}$$

where $\mathbf{U} \in \mathbb{R}^{(\sum_{\ell=1}^{L} N_\ell)\times R}$, $\mathbf{\Sigma} \in \mathbb{R}^{R\times R}$, and $\mathbf{V} \in \mathbb{R}^{bd\times R}$, with $R = \min\{\sum_{\ell=1}^{L} N_\ell, bd\}$. We select the first S right singular vectors and reshape them into $b \times d$ components for initialization:

$$\mathbf{B}_j = \text{reshape}_{b\times d}(\mathbf{v}_j), \qquad j = 1, \ldots, S, \tag{8}$$

where $\mathbf{v}_j$ denotes the j-th column of $\mathbf{V}$. This initializes the shared bank with the dominant gradient directions observed across blocks and layers. The SVD is performed only once for each projection type during initialization and its practical overhead is reported in the Section 4.4.

Block selection. For each block, we compute the cosine similarity between its rescaled gradient and each component in the shared bank of its projection type, where both matrices are flattened before computing the similarity: $\text{Sim}_{\ell,i,j} = \cos\left(\widetilde{\mathbf{G}}_{\ell,i}, \mathbf{B}_j\right)$. We select the K components with the largest absolute similarity scores and denote their indices by $\mathcal{S}_{\ell,i}$. The signed similarity scores of the selected components are then ℓ_2-normalized to initialize their coefficients:

$$c_{\ell,i,j} = \frac{\text{Sim}_{\ell,i,j}}{\sqrt{\sum_{k\in\mathcal{S}_{\ell,i}} \text{Sim}^2_{\ell,i,k}}}, \qquad j \in \mathcal{S}_{\ell,i}. \tag{9}$$

The selection remains fixed during fine-tuning, while the coefficients and shared components are jointly optimized.

The complete initialization and fine-tuning procedure is summarized in Algorithm 1.

4 EXPERIMENTS

We evaluate BLOC from three complementary perspectives. First, we compare its performance with matched-budget PEFT baselines across multiple backbones and task families. Second, we examine the expressivity of the learned updates by studying performance across adaptation budgets and analyzing their effective ranks. Third, we investigate the role of cross-layer sharing through sharing granularity and component-reuse analyses. We further ablate initialization, support assignment, and composition sparsity to assess the contribution of individual design choices.

4.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate BLOC on commonsense reasoning, mathematical reasoning, and code generation. Commonsense evaluation uses BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), Social IQa (Sap et al., 2019), ARC-Challenge and ARC-Easy (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), HellaSwag (Zellers et al., 2019), and WinoGrande (Sakaguchi et al., 2020), reporting per-task accuracy and their macro-average. We further fine-tune on 100K MetaMathQA (Yu et al., 2023) examples for GSM8K (Cobbe et al., 2021), and on 100K CodeFeedback-Filtered-Instruction (Zheng et al., 2024) examples for HumanEval (Chen et al., 2021).

Models and baselines. We evaluate Llama-2-7B (Touvron et al., 2023), Llama-3-8B (Grattafiori et al., 2024), and Qwen3-4B (Yang et al., 2025). We compare BLOC with LoRA (Hu et al., 2022), DoRA (Liu et al., 2024a), HiRA (Huang et al., 2025), and VB-LoRA (Li et al., 2024). Trainable-parameter budgets are matched as closely as permitted by each parameterization, with the exact parameter ratios reported in the corresponding results. Unless otherwise specified, all ablation and analysis experiments are conducted on Llama-3-8B.

Implementation details. All methods adapt W_q, W_k, W_v, W_{up}, and W_{down} for three epochs. Unless otherwise specified, BLOC partitions each update into blocks of size $b = 64$ and composes each block from $K = 2$ components selected from a component bank shared across Transformer layers. We use $M = 8$ calibration mini-batches for initialization, $\beta = 0.5$ for gradient weighting, and a learning rate of 10^{-3}. Main results report mean $\pm$ sample standard deviation over three training seeds. Complete training configurations, parameter budgets, and ablation protocols are provided in Appendix A.1.

4.2 MAIN RESULTS

Tables 1 and 2 compare BLOC with matched-budget PEFT baselines across three backbones and multiple task families. On commonsense reasoning, BLOC achieves the highest macro-average on all three backbones, reaching 83.76%, 87.29%, and 86.82% on Llama-2-7B, Llama-3-8B, and Qwen3-4B, respectively, compared with 82.36%, 86.72%, and 85.40% for the strongest baselines. These results are obtained with comparable or slightly fewer trainable parameters. BLOC also performs strongly across individual tasks, indicating that the aggregate improvements are not limited to a small subset of benchmarks.

The advantage extends to mathematical reasoning and code generation. On GSM8K, BLOC achieves 61.33%, 78.59%, and 84.23% across the three backbones, compared with 54.95%, 75.27%, and 83.56% for the strongest baselines. On HumanEval, BLOC reaches 35.57%, 53.86%, and 71.54%, while the corresponding strongest baseline results are 23.98%, 51.22%, and 70.32%. The improvements are particularly pronounced on the Llama backbones, while BLOC also maintains the best results on Qwen3-4B. Overall, BLOC consistently improves matched-budget performance across commonsense, mathematical, and code reasoning tasks and across all three model backbones.

4.3 ANALYSIS

Budget scaling. Figure 4 evaluates BLOC under trainable-parameter budgets matched to LoRA ranks 4, 8, 16, and 32 on Llama-3-8B. BLOC achieves the best performance among the evaluated methods at every matched budget, including the most constrained rank-4-equivalent setting. Its

Table 1: Commonsense reasoning accuracy (%) under matched PEFT budgets. BLOC achieves the highest average accuracy across all three backbones. Params denotes the trainable-parameter ratio.

Method	Params (%)	BoolQ	PIQA	SIQA	ARC-c	ARC-e	OBQA	HellaS	WinoG	Avg.
					Llama-2-7B					
LoRA	0.8256	66.21 ± 0.61	82.71 ± 0.55	79.02 ± 0.58	72.95 ± 0.90	86.05 ± 0.02	80.26 ± 1.13	93.53 ± 0.37	84.23 ± 0.58	80.62 ± 0.36
DoRA	0.8384	66.87 ± 0.51	82.88 ± 0.45	79.10 ± 0.57	72.64 ± 1.08	86.00 ± 0.10	80.66 ± 0.50	93.64 ± 0.30	84.55 ± 0.27	80.79 ± 0.17
HiRA	0.8256	70.58 ± 0.31	85.13 ± 0.36	80.41 ± 0.06	74.27 ± 0.14	87.66 ± 0.02	83.09 ± 0.27	93.68 ± 0.14	84.17 ± 0.26	82.36 ± 0.02
VB-LoRA	0.8333	70.60 ± 0.36	84.13 ± 0.19	80.72 ± 0.34	71.28 ± 0.39	86.86 ± 0.09	84.11 ± 0.11	93.89 ± 0.09	84.77 ± 0.08	82.05 ± 0.05
BLOC	0.8108	74.31 ± 0.07	84.69 ± 0.07	82.05 ± 0.06	76.10 ± 0.07	87.85 ± 0.06	85.20 ± 0.08	94.00 ± 0.02	85.84 ± 0.07	**83.76 ± 0.06**
					Llama-3-8B					
LoRA	0.7002	71.87 ± 0.71	89.09 ± 0.33	81.59 ± 0.21	83.08 ± 0.80	92.72 ± 0.31	88.01 ± 0.40	96.15 ± 0.07	87.84 ± 0.23	86.29 ± 0.01
DoRA	0.7098	71.77 ± 0.38	88.82 ± 0.30	81.90 ± 0.56	83.31 ± 0.53	92.92 ± 0.26	88.17 ± 0.33	96.14 ± 0.05	87.98 ± 0.13	86.38 ± 0.08
HiRA	0.7002	75.11 ± 0.12	89.50 ± 0.14	82.09 ± 0.20	82.25 ± 0.35	92.55 ± 0.11	86.60 ± 0.65	96.13 ± 0.06	89.50 ± 0.26	86.72 ± 0.07
VB-LoRA	0.7066	71.57 ± 0.35	87.83 ± 0.14	81.20 ± 0.31	80.15 ± 0.15	91.20 ± 0.35	85.81 ± 0.60	95.66 ± 0.10	88.31 ± 0.30	85.21 ± 0.09
BLOC	0.6974	75.70 ± 0.51	89.73 ± 0.28	81.48 ± 0.09	83.86 ± 0.21	93.34 ± 0.19	89.03 ± 0.47	96.08 ± 0.34	89.08 ± 0.02	**87.29 ± 0.03**
					Qwen3-4B					
LoRA	0.5469	66.04 ± 0.63	86.15 ± 0.08	78.32 ± 0.29	88.53 ± 0.21	95.22 ± 0.02	87.50 ± 0.19	93.26 ± 0.16	82.64 ± 0.21	84.71 ± 0.04
DoRA	0.5632	65.81 ± 1.11	86.97 ± 0.20	77.76 ± 0.16	88.36 ± 0.21	95.64 ± 0.17	88.13 ± 0.79	93.45 ± 0.05	82.72 ± 0.08	84.86 ± 0.02
HiRA	0.5469	67.80 ± 0.31	87.16 ± 0.08	78.69 ± 0.14	88.53 ± 0.57	95.53 ± 0.30	88.64 ± 0.82	93.77 ± 0.09	82.87 ± 0.08	85.37 ± 0.02
VB-LoRA	0.5533	69.00 ± 0.48	86.90 ± 0.39	79.16 ± 0.33	87.07 ± 0.13	94.47 ± 0.13	89.05 ± 0.45	93.30 ± 0.09	84.26 ± 0.06	85.40 ± 0.03
BLOC	0.5476	72.29 ± 0.51	87.88 ± 0.01	80.00 ± 0.29	88.46 ± 0.45	95.75 ± 0.35	90.54 ± 0.47	94.23 ± 0.13	85.43 ± 0.63	**86.82 ± 0.17**

Table 2: GSM8K and HumanEval accuracy (%) under matched PEFT budgets. BLOC achieves the best performance across all three backbones. Params denotes the trainable-parameter ratio.

Model	Method	Params (%)	GSM8K	HumanEval
Llama-2-7B	LoRA	0.8256	45.13 ± 0.80	20.53 ± 0.93
	DoRA	0.8384	45.43 ± 0.95	18.90 ± 0.61
	HiRA	0.8256	54.95 ± 0.37	23.98 ± 0.70
	VB-LoRA	0.8333	48.96 ± 0.48	16.26 ± 1.27
	BLOC	0.8108	**61.33 ± 0.31**	**35.57 ± 0.35**
Llama-3-8B	LoRA	0.7002	71.04 ± 0.18	45.73 ± 1.61
	DoRA	0.7098	72.00 ± 0.05	45.53 ± 0.93
	HiRA	0.7002	75.27 ± 0.48	51.22 ± 2.20
	VB-LoRA	0.7066	71.50 ± 0.45	48.58 ± 1.86
	BLOC	0.6974	**78.59 ± 0.24**	**53.86 ± 0.29**
Qwen3-4B	LoRA	0.5469	81.97 ± 0.80	69.31 ± 0.35
	DoRA	0.5632	82.00 ± 0.38	70.32 ± 0.35
	HiRA	0.5469	83.56 ± 0.53	69.92 ± 0.93
	VB-LoRA	0.5533	66.47 ± 0.76	67.28 ± 0.35
	BLOC	0.5476	**84.23 ± 0.38**	**71.54 ± 0.93**

average accuracy further increases from 86.16% to 87.27% as the available adaptation budget grows. These results show that the advantage of BLOC is preserved under tight parameter budgets while additional component capacity can be effectively utilized as more trainable parameters become available.

Spectral dimensionality. We further examine the singular-value spectra of the learned BLOC updates. We report effective rank together with $R_{90\%}$ and $R_{99\%}$, which measure the numbers of singular directions needed to retain 90% and 99% of the spectral energy, respectively. Detailed definitions are provided in Appendix B. Under a trainable-parameter budget matched to LoRA with rank 32, BLOC achieves effective ranks of 313.0–1155.5, with $R_{90\%}$ ranging from 374.5 to 1257.2 and $R_{99\%}$ from 742.2 to 2240.0. These results show that substantial spectral energy is distributed across hundreds to thousands of directions rather than arising only from negligible singular-value tails. The broader spectra of the FFN projections further indicate that the realized update dimensionality varies across projection types without being explicitly prescribed.

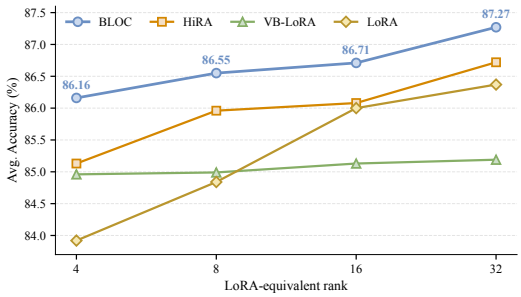


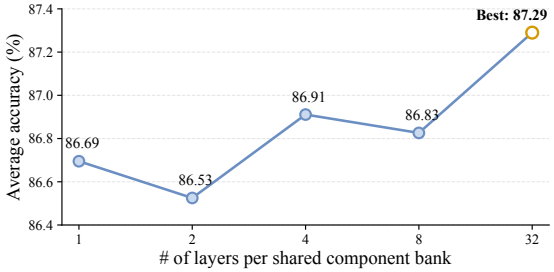
Figure 4: **Budget scaling under matched trainable-parameter budgets.** Across all LoRA-equivalent ranks, BLOC consistently achieves the best performance and continues to improve as the available parameter budget increases.

Table 3: **Spectral dimensionality of BLOC updates.** Layer-averaged effective rank and numbers of singular directions required to retain 90% and 99% of the spectral energy under a budget matched to LoRA rank 32.

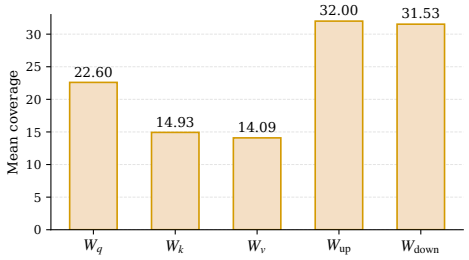
Projection	Eff. rank	R@90%	R@99%
W_q	423.2	528.3	1210.8
W_k	313.0	374.5	742.2
W_v	386.5	451.4	793.6
W_{up}	1115.3	1257.2	2240.0
W_{down}	1155.5	1244.4	2201.3

Sharing granularity. We first examine whether broader cross-layer sharing is beneficial under a fixed parameter budget. We vary the number of Transformer layers sharing each component bank from 1 to 32 while keeping the total component and trainable-parameter budgets fixed. A group size of 1 corresponds to no cross-layer sharing, whereas 32 corresponds to the default global-sharing setting. As shown in Figure 5(a), global sharing achieves the best performance, while restricting each bank to smaller groups of layers provides no improvement. Under the same budget, partitioning layers into smaller groups also fragments the available component capacity across multiple banks. In contrast, global sharing allows every layer to draw from the full component set while retaining layer-specific selections and coefficients.

Cross-layer component reuse. Having established the benefit of global sharing, we next examine how the shared component capacity is actually utilized across layers. In the Llama-3-8B main configuration, all 43 components are selected by multiple layers for W_q, W_k, W_{up}, and W_{down}, while 35 of 43 are reused for W_v. Figure 5(b) further shows that individual components typically span many layers: mean coverage reaches 32.00 layers for W_{up} and 31.53 for W_{down}, and ranges from 14.09 to 22.60 for the attention projections. These results indicate that the globally shared banks are broadly reused across layers rather than behaving as collections of effectively layer-specific components. The support heatmaps in Appendix C further show that different layers retain distinct component combinations despite drawing from the same shared bank.



(a) Effect of cross-layer sharing granularity.



(b) Cross-layer component reuse.

Figure 5: **Analysis of cross-layer sharing in BLOC.** (a) We vary the number of layers sharing each component bank while keeping the total trainable-parameter budget fixed; the default global-sharing setting with all 32 layers sharing the same projection-specific bank achieves the best performance. (b) We measure cross-layer component reuse as the mean number of layers using each component. Components are broadly reused across layers, with particularly high coverage for the FFN projections.

Together, these analyses show that BLOC can form high rank updates while extensively reusing trainable block components across Transformer layers. Combined with the budget-scaling results, they support blockwise composition and cross-layer sharing as complementary mechanisms for expressive, parameter-efficient adaptation.

4.4 Ablation Study

Initialization and assignment. Table 4 examines the effects of task-informed initialization and block component assignment. We first replace the SVD-based component initialization with a non-task-informed alternative while keeping the remaining BLOC parameterization unchanged. We then replace the cosine-guided Top-K support assignment with random assignment under the same shared compositional structure. Both variants remain above the matched LoRA baseline, suggesting that the blockwise compositional parameterization itself accounts for much of the improvement. Full BLOC further increases average accuracy by 0.33% over the variant without SVD initialization and by 0.25% over random Top-2 assignment, showing additional benefits from task-informed initialization and block component selection.

Composition sparsity. Figure 6 varies the number of selected block component per block while keeping the component-bank size fixed. Increasing K from 1 to 2 improves average accuracy from 86.77% to 87.27%, showing that combining multiple shared components is beneficial. Further increasing K yields no additional improvement, indicating that BLOC benefits from sparse composition without requiring dense component supports. We therefore use $K = 2$ as the default setting.

Table 4: Llama-3-8B component ablation. Values are mean ± std over three training or assignment seeds.

Setting	Avg. accuracy (%)
No SVD initialization	86.96 ± 0.13
Random Top-2	87.04 ± 0.05
Full BLOC	**87.29 ± 0.03**

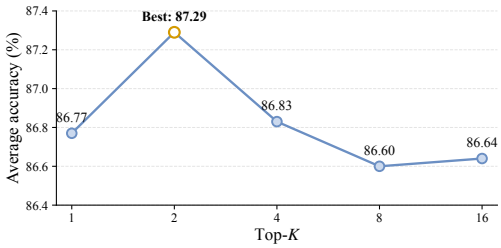


Figure 6: **Sensitivity to the number of selected components** K on Llama-3-8B commonsense reasoning tasks.

Block-size sensitivity. We examine the sensitivity of BLOC to the block size b, which controls the granularity of local composition. As shown in Figure 7, average accuracy remains relatively stable across $b \in \{32, 64, 128, 256\}$, ranging from 86.77% to 87.31% in these controlled runs. In contrast, initialization cost is much more sensitive to block size. In particular, increasing b from 32 to 64 greatly reduces initialization time while retaining nearly the same accuracy (87.31% versus 87.28%). Increasing b further provides only modest additional cost reductions without improving performance. We therefore use $b = 64$ as the default, indicating that BLOC is relatively robust to the block granularity while this choice avoids most of the initialization overhead of smaller blocks.

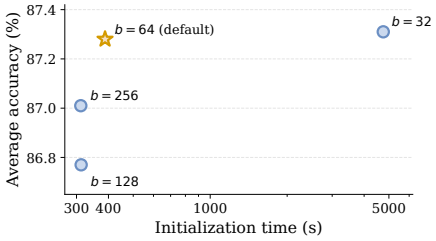


Figure 7: Block-size trade-off on Llama-3-8B. Initialization time is shown on a logarithmic scale. Each setting corresponds to one controlled run.

5 Conclusion

We introduced **BLOC**, a blockwise compositional PEFT method that constructs weight updates from trainable block components as an alternative to conventional multiplication-based parameterizations. Cross-layer component sharing maintains parameter efficiency, while gradient-guided calibration provides a task-informed initialization. Across three backbones and diverse reasoning tasks, BLOC achieves the strongest aggregate performance among the evaluated methods under matched parameter budgets. Further analyses reveal broad and heterogeneous spectral dimensionality, effective use of additional component capacity, and extensive cross-layer component reuse. Overall, these results demonstrate the effectiveness of blockwise composition for expressive and parameter-efficient model adaptation.

REFERENCES

Paul Albert, Frederic Z. Zhang, Hemanth Saratchandran, Cristian Rodriguez-Opazo, Anton van den Hengel, and Ehsan Abbasnejad. RandloRA: Full rank parameter-efficient fine-tuning of large models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Hn5eoTunHN.

Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 2005.

Daniel Bershatsky, Daria Cherniuk, Talgat Daulbaev, Aleksandr Mikhalev, and Ivan Oseledets. LoTR: Low tensor rank weight adaptation. *arXiv preprint arXiv:2402.01376*, 2024.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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

Emily Dinan, Varvara Logacheva, Valentin Malykh, Alexander Miller, Kurt Shuster, Jack Urbanek, Douwe Kiela, Arthur Szlam, Iulian Serban, Ryan Lowe, Shrimai Prabhumoye, Alan W Black, Alexander Rudnicky, Jason Williams, Joelle Pineau, Mikhail Burtsev, and Jason Weston. The second conversational intelligence challenge (convai2), 2019. URL https://arxiv.org/abs/1902.00098.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=NEv8YqBROO.

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

Qiushi Huang, Tom Ko, Zhan Zhuang, Lilian Tang, and Yu Zhang. HiRA: Parameter-efficient hadamard high-rank adaptation for large language models. In *International Conference on Learning Representations*, 2025.

Ting Jiang, Shaohan Huang, Shengyue Luo, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, Qi Zhang, Deqing Wang, and Fuzhen Zhuang. Mora: High-rank updating for parameter-efficient fine-tuning, 2024. URL https://arxiv.org/abs/2405.12130.

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

Yang Li, Shaobo Han, and Shihao Ji. Vb-lora: Extreme parameter efficient fine-tuning with vector banks. In *Advances in Neural Information Processing Systems*, 2024.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, 2004.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *International Conference on Machine Learning*, 2024a.

Zequan Liu, Jiawen Lyn, Wei Zhu, Xing Tian, and Yvette Graham. ALoRA: Allocating low-rank adaptation for fine-tuning large language models. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 622–641, Mexico City, Mexico, June 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.35. URL https://aclanthology.org/2024.naacl-long.35/.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models, 2025. URL https://arxiv.org/abs/2404.02948.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of EMNLP*, 2018.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of ACL*, 2002.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQA: Commonsense reasoning about social interactions. In *Proceedings of EMNLP-IJCNLP*, 2019.

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

Shaowen Wang, Linxi Yu, and Jian Li. Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems*, 37:54905–54931, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of ACL*, 2019.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=lq62uWRJjiY.

Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. AutoLoRA: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5048–5060, Mexico City, Mexico, June 2024. Association for Computational

Linguistics. doi: 10.18653/v1/2024.naacl-long.282. URL https://aclanthology.org/2024.naacl-long.282/.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations*, 2020.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*, 2024.

LIMITATIONS

Our evaluation focuses on a limited set of language models and reasoning tasks, leaving broader model families and application domains for future study. In addition, although we analyze several key design choices, exhaustive hyperparameter studies across all backbones and tasks are beyond the scope of this work. Future work could further evaluate the robustness and generality of BLOC under a wider range of model scales, tasks, and training settings.

AI USE STATEMENT

Generative AI tools were used to assist with LaTeX organization, language editing, arithmetic verification, and table formatting. The authors reviewed the generated text and calculations and take responsibility for the final content, claims, and artifacts.

A IMPLEMENTATION AND REPRODUCIBILITY DETAILS

A.1 TRAINING SETUP

All methods adapt W_q, W_k, W_v, W_{up}, and W_{down}. Unless otherwise stated, all models are trained for three epochs using AdamW with cosine learning-rate scheduling and a warmup ratio of 0.1. We use a per-device batch size of 16 and a maximum sequence length of 2,048. All experiments are conducted on NVIDIA A800 GPUs. For commonsense reasoning, we follow the training and evaluation setup described in the main text. For mathematical and code reasoning, models are trained on MetaMathQA-100K and CodeFeedback-100K, respectively.

A.2 METHOD-SPECIFIC CONFIGURATIONS

All baseline methods use zero adapter dropout. LoRA, DoRA, and HiRA use $r = 32$ for Llama-2-7B and Llama-3-8B, and $r = 16$ for Qwen3-4B. Table 5 summarizes the remaining method-specific hyperparameters.

Table 5: Method-specific hyperparameters for the baseline methods.

Method	Alpha / scale	Dropout	Learning rate(s)
LoRA	$\alpha = 2r$	0.0	10^{-5}
DoRA	$\alpha = 2r$	0.0	10^{-5}
HiRA	scale $= 1.0$	0.0	10^{-3}
VB-LoRA	–	0.0	10^{-3} / 10^{-2}

HiRA initializes its two trainable factors using Kaiming and zero initialization, respectively. VB-LoRA uses vectors of length 256, Top-$K = 2$, and no bias, with learning rates of 10^{-3} for the vector bank and 10^{-2} for the routing logits. For Llama-2-7B and Llama-3-8B, VB-LoRA uses rank 32 with 2,048 vectors; for Qwen3-4B, it uses rank 4 with 1,024 vectors.

Table 6 reports the BLOC configurations used in the main experiments. Here, b is the block size, S is the number of components in each projection-specific shared bank, K is the number of selected components per block, and β is the gradient-scaling exponent. The LoRA-equivalent rank denotes the matched parameter budget rather than a rank constraint on BLOC.

Table 6: Model-specific BLOC configurations used in the main experiments.

Backbone	LoRA-equiv. r	b	S per bank	K	β	Params. (%)
Llama-2-7B	32	64	42	2	0.5	0.8108
Llama-3-8B	32	64	43	2	0.5	0.6974
Qwen3-4B	16	64	26	2	0.5	0.5476

A.3 MATCHED-BUDGET SCALING

For the Llama-3-8B budget study, we compare LoRA, HiRA, VB-LoRA, and BLOC under approximately matched trainable-parameter budgets corresponding to LoRA ranks $r \in \{4, 8, 16, 32\}$, referred to as *LoRA-equivalent ranks*. LoRA uses $\alpha = 2r$ at each budget. For VB-LoRA, we keep its native rank at 4 and vary the vector-bank size; for BLOC, we vary the number of components per projection-specific bank as $S \in \{5, 11, 22, 43\}$.

Table 7: VB-LoRA configurations on Llama-3-8B for the matched-budget study.

LoRA-equiv. rank	# vectors	Trainable params.	Params. (%)
4	256	7.143M	0.0889
8	512	14.287M	0.1776
16	1024	28.574M	0.3546
32	2048	57.147M	0.7066

B RANK AND SPECTRAL METRICS

We compute all rank statistics from the reconstructed weight updates $\Delta\mathbf{W}_\ell$ after fine-tuning. For each layer and projection type, let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m \geq 0$ denote the singular values of $\Delta\mathbf{W}_\ell$.

For the layer-wise rank visualization in Figure 1, we compute the rank as the number of singular values greater than 10^{-4}.

For the spectral analysis, we normalize the squared singular values as $\pi_i = \frac{\sigma_i^2}{\sum_{j=1}^m \sigma_j^2}$,. The effective rank is then defined from the entropy of this spectral-energy distribution:

$$r_{\text{eff}} = \exp\left(-\sum_{i=1}^{m} \pi_i \log \pi_i\right). \tag{10}$$

This quantity characterizes the effective dimensionality of the update. A larger effective rank indicates that the spectral energy is distributed across more singular directions. Using squared singular values makes this distribution correspond to the Frobenius-norm energy carried by each direction and reduces the influence of very small singular values.

We additionally define the cumulative spectral energy of the leading r singular directions as

$$C(r) = \frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{j=1}^{m} \sigma_j^2}. \tag{11}$$

For an energy threshold τ, we compute

$$R_\tau = \min\{r : C(r) \geq \tau\}. \tag{12}$$

We report $R_{90\%}$ and $R_{99\%}$ using $\tau = 0.90$ and 0.99, respectively. All statistics in Table 3 are computed for each layer separately and then averaged across layers for each projection type.

C CROSS-LAYER SHARING ANALYSIS

We further analyze cross-layer component sharing from two perspectives: whether broader sharing improves the use of a fixed parameter budget, and whether the shared components are actually reused across Transformer layers.

Sharing granularity. We vary the scope of cross-layer sharing while keeping the total trainable-parameter budget fixed. Specifically, the 32 Transformer layers are divided into G contiguous groups, and the 43 components of each projection-specific bank are partitioned as evenly as possible among these groups. Components assigned to a group can only be selected by blocks within that group. Thus,

$G = 1$ corresponds to the default setting in which all 32 layers share the full bank of 43 components, whereas increasing G progressively restricts the sharing scope.

Table 8 reports the resulting allocations. "Component allocation" describes how the 43 components are distributed across the G groups. For example, $11 \times 3 + 10 \times 1$ indicates that three groups receive 11 components each and one group receives 10. "Effective K" denotes the actual support size used by each block under the corresponding allocation, and Δ denotes the change in average accuracy relative to the fully shared setting ($G = 1$).

Table 8: Effect of sharing granularity on the eight Llama-3-8B commonsense datasets under the same 0.6974% trainable-parameter budget. Accuracy is reported as mean $\pm$ sample standard deviation over three runs.

G	Layers/group	Component allocation	Effective K	Avg. (%)	Δ
1	32	43	2	87.290 ± 0.030	0.000
4	8	$11\times3 + 10\times1$	2	86.826 ± 0.102	-0.464
8	4	$6\times3 + 5\times5$	2	86.911 ± 0.161	-0.379
16	2	$3\times11 + 2\times5$	2	86.525 ± 0.069	-0.765
32	1	$2\times11 + 1\times21$	2 (11), 1 (21)	86.695 ± 0.239	-0.595

For $G = 32$, eleven layers receive two components and the remaining twenty-one receive only one, making the latter effectively Top-1. Among the tested allocations, the fully shared setting ($G = 1$) achieves the highest average performance. This result suggests that allowing more layers to access the same component bank makes more effective use of the available component capacity under a fixed parameter budget.

Component reuse. We further examine how components are reused across layers. Figure 8 shows the component selections for each projection type. Each row corresponds to a Transformer layer, each column to a component in the shared bank, and each cell records how many blocks in that layer select the component. When the same component is selected by multiple layers, it appears repeatedly in the same column, indicating cross-layer reuse. At the same time, different layers show different selection patterns. This suggests that BLOC can reuse the same component bank across layers while still allowing each layer to form its own local composition.

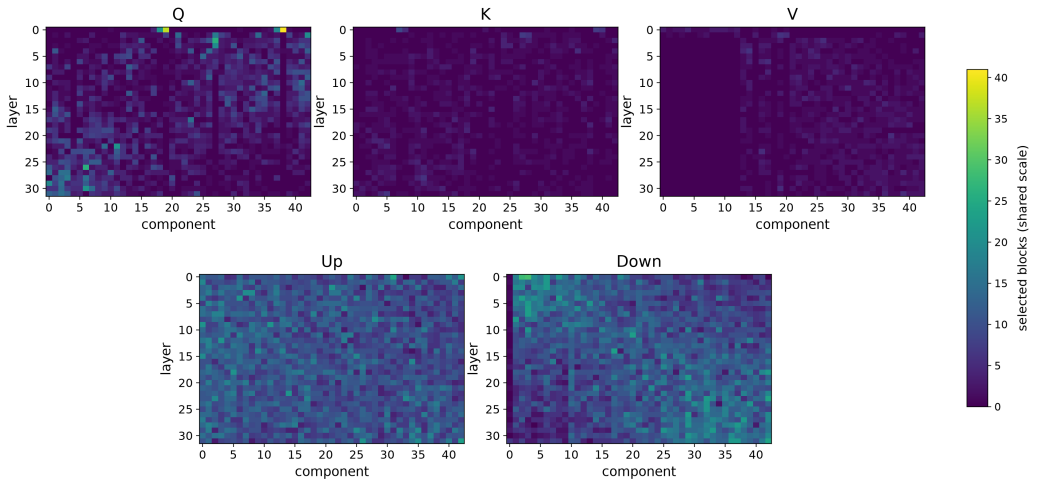


Figure 8: **Layer–component selection patterns.** Each cell shows the number of blocks in a layer that select a component. Repeated selections of the same components across layers indicate cross-layer reuse, while different row patterns reflect layer-specific compositions.

D TASK-INFORMED INITIALIZATION AND ASSIGNMENT ABLATIONS

Gradient–component alignment. We analyze how well the components selected during calibration align with their corresponding task-gradient blocks. Following the notation in the main text, the similarity between the rescaled gradient block $\widetilde{\mathbf{G}}_{\ell,i}$ and component $\mathbf{B}_j$ is

$$\mathrm{Sim}_{\ell,i,j} = \cos\left(\widetilde{\mathbf{G}}_{\ell,i}, \mathbf{B}_j\right).$$

Since component selection is based on the magnitude of this similarity, we define the alignment of block (ℓ, i) as the mean absolute cosine similarity to its selected components:

$$A_{\ell,i} = \frac{1}{K} \sum_{j \in \mathcal{S}_{\ell,i}} \left|\mathrm{Sim}_{\ell,i,j}\right|. \tag{13}$$

To examine how alignment varies with gradient magnitude, we sort the calibration blocks according to their original gradient norms $\|\mathbf{G}_{\ell,i}\|_F$ and divide them into four equal-sized quartiles, Q1–Q4, from the smallest to the largest norms. The alignment of quartile Q_q is then

$$A(Q_q) = \frac{1}{|Q_q|} \sum_{(\ell,i) \in Q_q} A_{\ell,i}. \tag{14}$$

Each quartile contains 4,352 calibration gradient blocks.

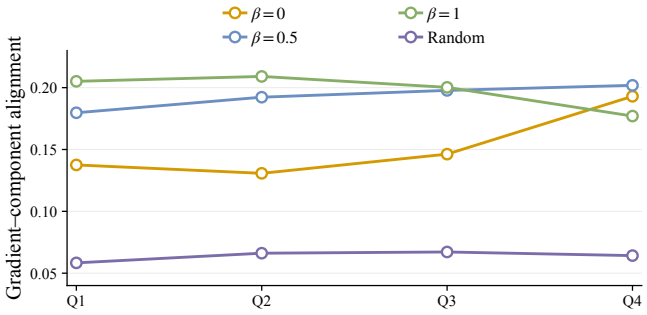


Figure 9: Gradient–component alignment across gradient-norm quartiles on Llama-3-8B. Each quartile contains 4,352 calibration gradient blocks.

Table 9: Gradient–component alignment by gradient-norm quartile on Llama-3-8B. Random Top-2 values are mean ± sample standard deviation over three assignment seeds.

Configuration	Q1	Q2	Q3	Q4	$\Delta_{\mathrm{Q4-Q1}}$	ρ
$\beta = 0$	0.13752	0.13074	0.14624	0.19301	+0.05549	+0.27750
$\beta = 0.5$	0.17969	0.19228	0.19777	0.20188	+0.02219	+0.17178
$\beta = 1$	0.20511	0.20907	0.20025	0.17703	−0.02808	−0.14943
Random	0.05834 ± 0.00041	0.06615 ± 0.00027	0.06714 ± 0.00087	0.06421 ± 0.00019	$+0.00587 \pm 0.00045$	$+0.05175 \pm 0.00761$

The results reveal two effects of the task-informed initialization. First, all SVD-based configurations achieve substantially higher alignment than Random Top-2 across all quartiles, indicating that cosine-guided selection consistently assigns components whose directions better match the calibration gradients. Second, β controls how this alignment is distributed across gradient magnitudes. With $\beta = 0$, alignment is biased toward high-norm blocks, as reflected by the positive $\Delta_{\mathrm{Q4-Q1}}$ and ρ. In contrast, $\beta = 1$ favors lower-norm blocks and reverses this trend. The intermediate setting $\beta = 0.5$ maintains relatively high alignment across all four quartiles while substantially reducing the imbalance between low- and high-norm gradients. This provides a task-informed motivation for using $\beta = 0.5$ as the default scaling exponent.

Component usage. We additionally examine whether cosine-guided assignment causes the selected supports to collapse onto a small subset of the component bank. Table 10 reports the distribution of component usage after initialization. All configurations activate all 43 components, and no single component accounts for more than 3.5% of the selections, indicating that the task-informed assignment does not collapse onto a small number of components. As expected, Random Top-2 produces a more uniform usage distribution, whereas cosine-guided assignment introduces moderate task-dependent variation in component usage.

Table 10: Component-use statistics after initialization. Random Top-2 values are mean ± sample standard deviation over three assignment seeds.

Configuration	Usage entropy	Gini	Max. share	Active components
$\beta = 0$	0.96065	0.25652	3.50%	43 / 43
$\beta = 0.5$	0.97686	0.16363	3.26%	43 / 43
$\beta = 1$	0.99206	0.12750	3.33%	43 / 43
Random	0.99765 ± 0.00034	0.06521 ± 0.00500	$3.00\% \pm 0.06\%$	43 / 43

E TOP-K TRAINING DYNAMICS

Figure 10 shows the training-loss trajectories for the five Top-K configurations used in Figure 6. The training trajectories are similar across different K values during most of training, indicating that increasing the support size does not accelerate convergence. The curves separate more clearly at the late stage, where $K = 1$ and $K = 2$ reach lower training losses than larger supports, while $K = 16$ remains consistently higher. Thus, selecting more components per block does not provide an optimization advantage in the evaluated setting. Together with the downstream results in Figure 6, these observations support $K = 2$ as a sufficiently expressive yet sparse choice for BLOC.

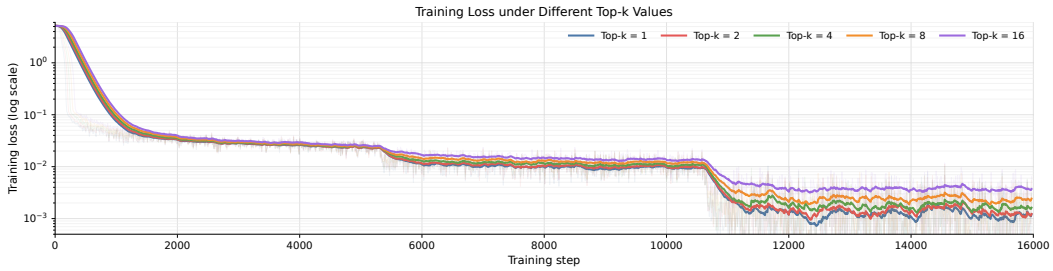


Figure 10: Training loss under different Top-K values. Each configuration uses the same axes; faint curves show raw loss and darker curves show the smoothed trajectory.

F TRAINING THROUGHPUT

Table 11 compares training throughput for LoRA, DoRA, HiRA, and BLOC on Llama-3-8B under a trainable-parameter budget matched to LoRA with rank 32. All methods use batch size 4. After 50 warm-up steps, we time 500 training steps for each of three random seeds (42, 43, and 44); values are reported as mean ± sample standard deviation.

Table 11: Training throughput on Llama-3-8B under the LoRA-rank-32-equivalent budget (batch size 4). Each method is timed for 500 steps after 50 warm-up steps; values are mean ± sample standard deviation over three seeds.

Method	Steps/s	Samples/s	Time for 500 steps (s)
LoRA	2.7455 ± 0.0333	10.9818 ± 0.1332	182.137 ± 2.212
DoRA	1.2147 ± 0.0048	4.8586 ± 0.0192	411.644 ± 1.625
HiRA	1.6062 ± 0.0056	6.4250 ± 0.0224	311.287 ± 1.083
BLOC	2.3175 ± 0.0292	9.2701 ± 0.1169	215.769 ± 2.720

BLOC is slower than LoRA but substantially faster than HiRA and DoRA under this configuration, while its one-time initialization cost is reported separately above.

G Additional Dialogue Results on ConvAI2

Table 12 reports results on ConvAI2 (Dinan et al., 2019) using Llama-2-7B and Llama-3-8B. ConvAI2 consists of persona-grounded multi-turn dialogues. We adopt the self-persona setting, in which only the speaker's own persona is provided during generation. We report BLEU-4 (Papineni et al., 2002), BERTScore precision, recall, and F1 (Zhang et al., 2020), METEOR (Banerjee & Lavie, 2005), and ROUGE-L (Lin, 2004). Avg. denotes their unweighted arithmetic mean.

At comparable parameter ratios, BLOC obtains the highest aggregate dialogue score on both backbones. The result extends the main reasoning evaluation to a generation setting, but is reported as supplementary evidence because it does not include the full set of main-table baselines.

Table 12: Results on ConvAI2 (%). Params. denotes the trainable parameter ratio. Avg. is the unweighted mean of the six generation metrics.

Method	Params. (%)	BLEU-4	BERT-F1	BERT-R	BERT-P	METEOR	ROUGE-L	Avg.
Llama-2-7B ($r = 32$)								
MoRA	0.8241	1.09	84.09	84.65	83.59	10.97	9.57	45.66
LoRA	0.8256	1.82	84.41	84.71	84.16	11.38	10.55	46.17
DoRA	0.8384	1.73	84.18	84.61	83.81	11.25	10.41	46.00
HiRA	0.8256	2.70	84.86	84.98	84.77	13.56	12.80	47.28
BLOC	0.8108	**4.33**	**86.64**	**86.41**	**86.88**	**12.67**	**22.53**	**49.91**
Llama-3-8B ($r = 32$)								
MoRA	0.6997	1.60	82.24	84.06	84.43	12.37	11.19	46.31
LoRA	0.7002	2.26	84.32	84.00	84.67	12.51	11.77	46.59
DoRA	0.7098	2.29	84.32	84.06	84.62	12.63	11.78	46.62
HiRA	0.7002	3.41	84.81	84.40	85.25	14.87	14.05	47.80
BLOC	0.6974	**5.45**	**87.29**	**86.84**	**87.77**	**16.15**	**22.67**	**51.02**