Breaking Information Islands in Sparse Tuning via Small-World Connectivity

Xianchao Guan[1,2] **Zijun Xiong**[1] **Yifeng Wang**[3] **Yubo Cui**[1]
Yaowei Wang[1,2] **Guo-Sen Xie**[4] **Xin Li**[2*] **Zheng Zhang**[1,2*]

[1]Harbin Institute of Technology, Shenzhen, China
[2]Pengcheng Laboratory, China
[3]Tsinghua University, China
[4]Nanjing University of Science and Technology, China

Abstract

Sparse tuning is widely used to adapt large language models due to its parameter efficiency. However, its effectiveness depends not only on the number of trainable parameters, but also on the feature-interaction topology of the sparse update's structural support. Limited or uneven connectivity in this support may isolate subsets of feature dimensions, forming information islands that restrict direct task-specific interactions through the trainable update. To address this issue, we propose **S**mall-**W**orld **I**nduced **F**ine-**T**uning (**SWIFT**), a sparse tuning framework inspired by small-world connectivity. SWIFT combines block-diagonal sparse updates for local feature mixing with a fixed deterministic scattering permutation that routes the mixed outputs across feature groups without additional trainable parameters. We further show theoretically that certain random sparse-update models and block-diagonal updates can exhibit information islands, whereas a sufficiently scattering permutation guarantees positive block-level expansion when the trainable block-diagonal factor consists of equal-sized, structurally dense square blocks. Extensive experiments on commonsense reasoning, natural language generation, and image classification demonstrate that SWIFT outperforms the evaluated parameter-efficient fine-tuning (PEFT) baselines on average and can match or surpass full fine-tuning. Implementation is available at: `https://github.com/xianchaoguan/SWIFT`.

1 Introduction

Large language models (LLMs) have shown strong capabilities across language understanding and generation tasks [Kuang et al., 2025, Axelsson and Skantze, 2023, Meng et al., 2026, 2025], accelerating the practical deployment of modern AI systems [Yang et al., 2024a, Jiang et al., 2026]. A common adaptation paradigm is to pre-train on large corpora and then fine-tune the model for specific tasks or domains [Zhang et al., 2023a, Parthasarathy et al., 2024]. However, as model scale increases, full fine-tuning becomes costly in computation, memory, and deployment, limiting its scalability in practice [Zhao et al., 2024a]. This has motivated parameter-efficient fine-tuning (PEFT), which aims to retain the adaptation ability of full fine-tuning while substantially reducing training and deployment costs [Kaplan et al., 2020, Zhao et al., 2024b, Yang et al., 2024b].

PEFT methods adapt large models by updating only a small set of parameters [Zhang et al., 2025]. Low-Rank Adaptation (LoRA) [Hu et al., 2022] is a representative approach, parameterizing weight updates with low-rank matrices while freezing pre-trained weights. Its variants further improve

*Corresponding authors: Xin Li(xinlihitsz@gmail.com) and Zheng Zhang(darrenzz219@gmail.com).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

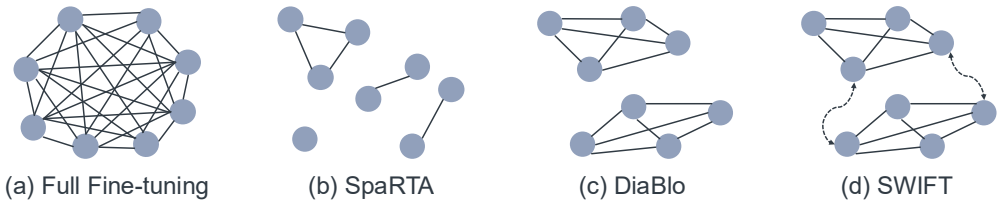


Figure 1: Schematic feature-interaction topologies of trainable updates in full fine-tuning, SpaRTA, DiaBlo, and SWIFT. SWIFT combines local feature mixing with permutation-based routing across feature groups.

expressiveness, initialization, and optimization behavior [Zhang et al., 2023b, Liu et al., 2024, Meng et al., 2024, Jiang et al., 2025]. However, low-rank methods are inherently constrained by rank bottlenecks, which may limit expressive capacity and downstream performance [Hayou et al., 2024, He et al., 2025a]. Sparse tuning offers an alternative by directly updating a discrete subset of pre-trained weights, maintaining parameter efficiency without imposing an explicit low-rank factorization [Song et al., 2023, Yang et al., 2024c, Wang et al., 2025]. Recent structured update parameterizations further suggest that the update structure itself is crucial for efficient adaptation [Tan et al., 2024, Gurses et al., 2025]. Yet how such structures shape information propagation across feature dimensions remains insufficiently understood.

From the perspective of forward computation, neural network inference can be viewed as information propagation in a high-dimensional feature space [Joshi, 2025]. Full fine-tuning allows dense weight updates, enabling direct task-specific interactions across feature dimensions, as illustrated in Figure 1(a). Sparse tuning, in contrast, updates only a subset of entries or structured components, inducing a sparse feature-interaction graph defined by the update's structurally trainable support. Limited or uneven connectivity in this graph may leave subsets of feature dimensions weakly connected or isolated, forming information islands that restrict direct task-specific interactions through the trainable update. Such islands arise in purely block-diagonal methods such as DiaBlo [Gurses et al., 2025], which preserve local interactions but lack trainable cross-block communication (Figure 1(c)). Irregular or random sparse methods such as SpaRTA [Aliaga et al., 2025] may also exhibit uneven feature-group coverage depending on the sampled support (Figure 1(b)).

This observation suggests that the effectiveness of sparse tuning depends not only on the number of trainable parameters, but also on how well the update supports global communication across feature dimensions. Small-world connectivity provides a natural principle for this goal, combining strong local interactions with sparse long-range shortcuts to enable efficient information exchange without dense connectivity [Lachgar and Achahbar, 2024, Morais and Interian, 2025]. In this context, an effective sparse update should combine local feature mixing with reliable access to distant feature groups, motivating designs that integrate structured local mixing with distributed long-range routing.

In this paper, we propose **S**mall-**W**orld **I**nduced **F**ine-**T**uning (**SWIFT**), a sparse tuning framework inspired by small-world connectivity (Figure 1(d)). SWIFT uses block-diagonal sparse updates for local feature mixing and applies a fixed deterministic scattering permutation to route the mixed outputs across feature groups without additional trainable parameters. Theoretically, we show that certain random sparse-update models and block-diagonal updates can exhibit information islands, whereas a sufficiently scattering permutation guarantees positive block-level expansion when the trainable block-diagonal factor consists of equal-sized, structurally dense square blocks. Extensive experiments on commonsense reasoning, natural language generation, and image classification demonstrate that SWIFT outperforms the evaluated PEFT baselines on average and can match or surpass full fine-tuning.

2 Related Work

2.1 Low-Rank Adaptation

Low-rank adaptation is a widely used PEFT paradigm due to its simplicity and strong empirical performance. LoRA [Hu et al., 2022] enables efficient adaptation by injecting trainable low-rank

matrices while freezing pre-trained weights. Subsequent work improves its expressiveness and optimization through better parameterization, initialization, and architectural design, including DoRA [Liu et al., 2024], PiSSA [Meng et al., 2024], HydraLoRA [Tian et al., 2024], and GOAT [Fan et al., 2025]. However, low-rank factorization restricts the rank of the parameterized update, which may limit the ability to model complex cross-dimensional interactions as adaptation demands increase [Zeng and Lee, 2023, Hayou et al., 2024].

2.2 Sparse Fine-Tuning

Sparse fine-tuning directly updates a subset of pre-trained weights, maintaining parameter efficiency without imposing an explicit low-rank factorization [Song et al., 2023, Yang et al., 2024c, Wang et al., 2025]. Existing methods differ in how the sparse update is constructed. Data-driven approaches select task-relevant parameters based on importance signals, such as SMT [He et al., 2025b] and SpIEL [Ansell et al., 2024]. In contrast, SpaRTA [Aliaga et al., 2025] randomly selects a fixed subset of parameters for adaptation. S^2FT [Yang et al., 2024c] selects attention heads and FFN channels, then co-permutes coupled weight matrices to organize the selected components into dense submatrices for efficient updates. Related structured parameterizations further highlight the importance of update design: MoRe [Tan et al., 2024] uses two trainable block-diagonal factors and two fixed permutations in a Monarch-style parameterization, while DiaBlo [Gurses et al., 2025] directly updates dense diagonal blocks.

SWIFT combines a single trainable block-diagonal factor with a fixed deterministic scattering permutation, yielding $\Delta W = BP$. This composition routes locally mixed outputs across feature groups without additional trainable parameters. Alongside this update design, we analyze how an explicit scattering condition controls remote-group coverage and expansion under stated structural assumptions.

3 Method

We present SWIFT, which combines block-diagonal updates with a fixed deterministic scattering permutation to route locally mixed outputs across feature groups. We then introduce Minimum Remote Group Coverage (MRGC), a method-agnostic measure of connectivity in the structural support of trainable updates, focusing on worst-case access to remote groups. Finally, we provide a graph-theoretic analysis of information islands and establish coverage and expansion guarantees under explicit structural and scattering assumptions.

3.1 SWIFT Framework

SWIFT is applied to standard linear modules in Transformers, such as query/key/value projections and feed-forward network layers. Consider a frozen pre-trained linear weight $\mathbf{W} \in \mathbb{R}^{m_1 \times m_2}$ in such a module. Given an input activation $\mathbf{x} \in \mathbb{R}^{1 \times m_1}$, the original computation is

$$\mathbf{y} = \mathbf{x}\mathbf{W}. \tag{1}$$

SWIFT introduces a sparse trainable update that combines local feature mixing with routing across feature groups. Specifically, we parameterize the effective update as a block-diagonal trainable matrix followed by a fixed permutation:

$$\mathbf{y} = \mathbf{x}\mathbf{W} + \mathbf{x}\mathbf{B}\mathbf{P}, \tag{2}$$

where $\mathbf{B} \in \mathbb{R}^{m_1 \times m_2}$ is trainable and block diagonal, and $\mathbf{P} \in \mathbb{R}^{m_2 \times m_2}$ is fixed throughout fine-tuning. Within each adapted linear module, only $\mathbf{B}$ is trainable. We initialize $\mathbf{B}$ to zero, preserving the original module's output at initialization. The block structure of $\mathbf{B}$ provides local feature mixing, while $\mathbf{P}$ routes the mixed outputs across feature groups. Thus, the effective update $\mathbf{BP}$ introduces cross-group interactions without increasing the number of trainable parameters, as illustrated in Figure 2.

Grouping: block-diagonal sparse updates. SWIFT first partitions the input and output feature dimensions of a linear module into $N > 1$ groups. When N divides both m_1 and m_2, let $d_1 = m_1/N$ and $d_2 = m_2/N$. We divide $\mathbf{B}$ into $N \times N$ blocks $\mathbf{B}_{ij} \in \mathbb{R}^{d_1 \times d_2}$ and enforce the block-diagonal

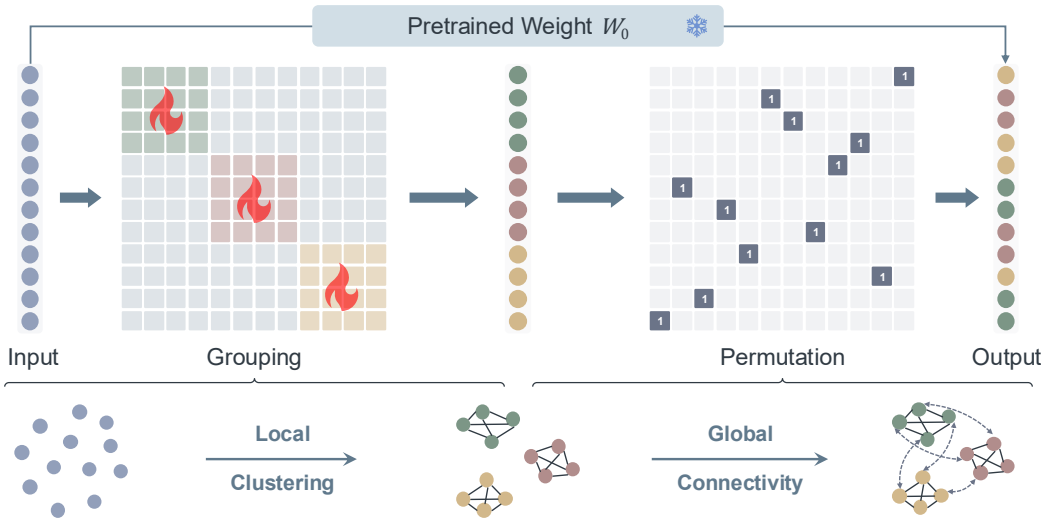


Figure 2: Overview of the SWIFT framework. SWIFT combines trainable block-diagonal updates for local feature mixing with a fixed deterministic scattering permutation that routes the mixed outputs across feature groups. The pre-trained weights remain frozen.

structure

$$\mathbf{B} = \begin{pmatrix} \mathbf{B}_{11} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_{22} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{B}_{NN} \end{pmatrix}. \tag{3}$$

During fine-tuning, all entries within the diagonal blocks $\{\mathbf{B}_{ii}\}_{i=1}^{N}$ are trainable, while the off-diagonal blocks remain zero. The number of trainable parameters is $Nd_1d_2 = m_1m_2/N$. This structure concentrates the trainable budget on dense within-group interactions, providing efficient local adaptation. By itself, a block-diagonal update has no trainable interactions between differently indexed input and output groups.

Permutation: long-range routing. To enable cross-group interactions without introducing additional trainable parameters, SWIFT composes the block-diagonal update with a fixed permutation $\mathbf{P}$. Right multiplication by $\mathbf{P}$ reorders the output coordinates of $\mathbf{xB}$, dispersing each block's mixed outputs across feature groups. Consequently, the effective update $\mathbf{BP}$ connects input groups to dispersed output coordinates under the same parameter budget. The permutation is determined by the layer dimensions and block number, remains fixed throughout training, and requires no optimizer state.

Efficient implementation. In practice, $\mathbf{P}$ is implemented by index reordering and is never explicitly materialized as a dense matrix. We generate its index array using a reshape–transpose–reshape procedure: output indices are reshaped into an $N \times d_2$ grid, transposed, and flattened. The resulting array is used to gather output coordinates. Writing $\pi(j)$ for the pre-permutation coordinate selected for output coordinate j, our convention is

$$(\mathbf{BP})_{ij} = \mathbf{B}_{i,\pi(j)}. \tag{4}$$

When N divides the output width, the number of coordinates routed from any one intermediate output group to any one final output group is either $\lfloor d_2/N \rfloor$ or $\lceil d_2/N \rceil$. Each group therefore reaches exactly $\min(N, d_2)$ target groups. Exact uniform scattering requires $N \mid d_2$, equivalently $N^2 \mid m_2$. Otherwise, the pairwise loads differ by at most one coordinate, with realized scattering bound

$$\epsilon_{\text{real}} = \frac{\lceil d_2/N \rceil}{d_2}. \tag{5}$$

We use $N = 64$ in all main experiments, for which the input and output widths of all adapted projections are divisible by N. For incompatible dimensions, the implementation pads the input and block dimensions to the next multiples of N, applies the permutation at the padded output width, and retains the first m_2 output coordinates. In this case, coverage and scattering are evaluated on the retained support after slicing.

3.2 Global Connectivity and Remote Group Coverage Analysis

The connectivity induced by the effective update is key to understanding sparse fine-tuning. We analyze its *structural support*: the set of positions at which the update is allowed to vary, including trainable entries initialized to zero. This support determines the possible direct task-specific interactions through the trainable update. A useful measure should therefore capture not only sparsity, but also whether each feature dimension has sufficient access to remote feature groups.

To this end, we introduce *Minimum Remote Group Coverage* (MRGC), a feature-wise worst-case measure of remote-group connectivity. Consider a square linear layer adapted as $\mathbf{y} = \mathbf{x}(\mathbf{W} + \Delta\mathbf{W})$, where $\mathbf{W}$ is frozen and $\Delta\mathbf{W}$ is the effective update. Let $\Omega_\Delta \subseteq \{1, \dots, d\} \times \{1, \dots, d\}$ denote the structural support of $\Delta\mathbf{W}$. Each pair $(i, j) \in \Omega_\Delta$ represents a possible direct trainable interaction from input feature dimension i to output feature dimension j.

Let $V = \{1, \dots, d\}$ be partitioned into $K > 1$ groups, $\mathcal{P} = \{V_1, \dots, V_K\}$, and let $g(i)$ denote the index of the group containing feature dimension i. For each source feature dimension i, we define the set of directly reachable remote target groups as

$$\mathcal{R}(i) = \{q \in \{1, \dots, K\} \setminus \{g(i)\} : \exists j \in V_q,\ (i, j) \in \Omega_\Delta\}. \tag{6}$$

The MRGC score is defined as

$$\mathrm{MRGC} = \min_{i \in V} \frac{|\mathcal{R}(i)|}{K - 1}. \tag{7}$$

MRGC measures the worst-case fraction of remote feature groups that each feature dimension can directly reach through the update. Poorly covered features may become bottlenecks for direct cross-group interactions. By excluding within-group interactions, MRGC does not reward purely local updates and, under a shared partition and matched density, provides a method-agnostic measure of remote-group coverage.

At $d = 4096$ and $K = 64$, SWIFT attains an MRGC of 1.00 at density $1/64$, compared with 0.00 for block-diagonal tuning and a mean of 0.41 over $1{,}000$ random masks at the same expected density; dense full fine-tuning also attains 1.00. Appendix C.2 provides the simulation setup and the corresponding group-coverage definition for rectangular layers.

3.3 Theory of Small-World Induced Fine-Tuning

We provide a graph-theoretic analysis of connectivity in the structural support of trainable updates. For square layers, we identify input and output coordinates with a common vertex set. Under this view, information islands are subsets of feature dimensions with no outgoing trainable interactions to their complement. We analyze how local mixing and permutation-based routing affect this structural connectivity.

Definition 3.1 (**Operator-Induced Interaction Graph**). Let $\Delta\mathbf{W} \in \mathbb{R}^{d \times d}$ be a trainable update with structural support Ω_Δ. Its *operator-induced interaction graph* is the directed graph $G_{\Delta\mathbf{W}} = (V, E)$ with $V = \{1, \dots, d\}$ and

$$E = \Omega_\Delta. \tag{8}$$

Under the row-vector convention $\mathbf{y} = \mathbf{x}(\mathbf{W} + \Delta\mathbf{W})$, an edge $(i, j) \in E$ represents a possible direct trainable interaction from input coordinate i to output coordinate j.

To quantify bottlenecks in this support graph, we use directed out-vertex expansion.

Definition 3.2 (**Directed Out-Vertex Expansion**). For a directed graph $G = (V, E)$ and a subset $S \subseteq V$, define

$$O(S) = \{j \in V : \exists i \in S,\ (i, j) \in E\}, \qquad \partial_{\mathrm{out}} S = O(S) \setminus S.$$

The coordinate-level out-vertex expansion is

$$h(G) \triangleq \min_{\substack{S \subseteq V \\ 1 \le |S| \le \lfloor d/2 \rfloor}} \frac{|\partial_{\text{out}} S|}{|S|}. \tag{9}$$

When $h(G) = 0$, there exists a nonempty subset of at most half the coordinates with no outgoing edges to its complement in the update-support graph.

Theorem 3.3 (**Connectivity Collapse under Independent Random Sparsification**). *Fix a constant $a > 0$. Consider a sequence of random structural supports on d vertices whose outgoing supports are sampled independently across vertices and whose expected average out-degree tends to a. Assume that*

$$\liminf_{d \to \infty} \min_{v \in \{1,\dots,d\}} \mathbb{P}\big(D_{\text{out}}(v) = 0\big) > 0.$$

Then the induced graph G_d contains zero-out-degree vertices with probability approaching 1, and

$$\lim_{d \to \infty} \mathbb{P}\big(h(G_d) = 0\big) = 1. \tag{10}$$

Theorem 3.3 identifies a failure mode of random sparse updates in the constant-expected-degree regime. For example, independent Bernoulli sampling with entry probability a/d satisfies the assumption because the zero-out-degree probability is $(1 - a/d)^d \to e^{-a} > 0$. A zero-out-degree vertex forms a singleton with no outgoing boundary. The fixed-density MRGC simulation above examines uneven group coverage at a different sparsity scale. A detailed proof is provided in Appendix B.2.

Theorem 3.4 (**Zero Expansion of Block-Diagonal Updates**). *Let $\mathbf{B} \in \mathbb{R}^{d \times d}$ have block-diagonal structural support with $k > 1$ equal-sized blocks. Then its induced graph satisfies*

$$h(G_{\mathbf{B}}) = 0. \tag{11}$$

Theorem 3.4 formalizes the connectivity bottleneck of purely block-diagonal updates. Each block has no outgoing edges to other blocks, and its size is at most $d/2$. Thus, a block forms a subset with zero outgoing boundary. A detailed proof is provided in Appendix B.3.

SWIFT addresses block isolation by composing $\mathbf{B}$ with a fixed permutation $\mathbf{P}$, which disperses locally mixed outputs across feature blocks. We capture this effect through a scattering condition.

Definition 3.5 (**ϵ-Uniform Scattering Permutation**). Partition $V = \{1, \dots, d\}$ into $k > 1$ equal-sized blocks $\mathcal{P} = \{V_r\}_{r=1}^k$ with block size $m = d/k$. A permutation $\pi : V \to V$ is *ϵ-uniformly scattering* with respect to this partition if

$$|\pi(V_r) \cap V_q| \le \epsilon m \quad \text{for all } r, q \in \{1, \dots, k\}, \qquad \epsilon \in [1/k, 1]. \tag{12}$$

We use the permutation-matrix convention $\mathbf{P}_{\pi(j),j} = 1$, so $(\mathbf{BP})_{ij} = \mathbf{B}_{i,\pi(j)}$. Smaller ϵ indicates more evenly dispersed block overlaps. Exact uniform scattering has $\epsilon = 1/k$ and requires $k \mid m$.

To characterize expansion across complete feature blocks, write $U_I = \bigcup_{r \in I} V_r$ and define the block-level out-vertex expansion as

$$h_{\mathcal{P}}(G) \triangleq \min_{\substack{I \subseteq \{1,\dots,k\} \\ 1 \le |I| \le \lfloor k/2 \rfloor}} \frac{|O(U_I) \setminus U_I|}{|U_I|}. \tag{13}$$

Because block unions form a restricted family of coordinate subsets,

$$h(G) \le h_{\mathcal{P}}(G).$$

MRGC measures per-vertex remote-group coverage and does not account for neighborhood overlap across vertices, so it has no general ordering with either expansion measure.

Theorem 3.6 (**Expansion of Small-World Induced Fine-Tuning**). *Let $\mathbf{B} \in \mathbb{R}^{d \times d}$ have $k > 1$ equal-sized, structurally dense diagonal blocks of size $m = d/k$, with*

$$\Omega_{\mathbf{B}} = \bigcup_{r=1}^{k} (V_r \times V_r).$$

Let $\mathbf{P}$ correspond to an ϵ-uniformly scattering permutation with respect to the same partition. If $\epsilon < 2/k$, then the structural support of the SWIFT update $\mathbf{BP}$ satisfies

$$h_{\mathcal{P}}(G_{\mathbf{BP}}) \ge 1 - \frac{k\epsilon}{2} > 0. \tag{14}$$

The same scattering condition separately yields MRGC and coordinate-level expansion bounds; their statements and proofs are provided in Appendix B.1 and Appendix B.4. Under exact uniform scattering, $\epsilon = 1/k$, the block-level lower bound is $1/2$.

Taken together, Theorems 3.3–3.6 identify distinct connectivity properties of sparse update supports and support combining local feature mixing with distributed cross-group routing under a sparse parameter budget. The expansion guarantees apply under the stated assumptions for square layers; rectangular projections are characterized through bipartite group coverage.

4 Experiments and Results

4.1 Baselines

We compare SWIFT with full fine-tuning; low-rank methods LoRA [Hu et al., 2022], DoRA [Liu et al., 2024], PiSSA [Meng et al., 2024], HydraLoRA [Tian et al., 2024], and GOAT [Fan et al., 2025]; and sparse or structured methods SpaRTA [Aliaga et al., 2025], S^2FT [Yang et al., 2024c], SMT [He et al., 2025b], SpIEL [Ansell et al., 2024], MoRe [Tan et al., 2024], and DiaBlo [Gurses et al., 2025]. Full fine-tuning updates all pre-trained parameters. Baseline descriptions and task-specific settings are provided in Appendices C.1 and A.2.

Table 1: Performance comparison of LLaMA3-8B on eight commonsense reasoning datasets. Task scores are accuracies (%). The best and second-best scores in each column are indicated in **bold** and underlined, respectively, including ties.

Method	Param. (%)	BoolQ	PIQA	SIQA	HellaS.	WinoG.	ARC-e	ARC-c	OBQA	Avg.
Full Fine-Tuning	100	73.90	86.20	79.10	93.10	85.80	88.10	78.20	84.00	83.55
Low-rank adaptation										
LoRA	0.70	74.58	88.41	80.81	95.31	85.87	89.60	80.12	85.40	85.01
DoRA	0.71	76.09	88.30	80.60	95.23	86.67	90.40	80.29	88.20	85.72
PiSSA	0.70	69.70	87.20	79.50	90.30	81.00	82.90	75.50	77.80	80.49
HydraLoRA	0.53	72.20	85.00	79.80	89.60	87.60	79.40	74.00	80.40	81.00
Sparse and structured adaptation										
SpIEL	0.70	72.10	83.60	80.00	91.80	85.40	91.20	76.80	80.80	82.71
S^2FT	0.70	75.00	89.00	80.70	**96.50**	88.00	92.50	83.40	87.80	86.61
MoRe	0.70	71.93	89.28	82.24	95.22	88.79	93.27	83.19	88.80	86.59
SMT	3.01	75.10	89.90	82.40	96.30	88.80	92.60	82.80	89.60	87.19
SpaRTA	0.44	75.60	**90.32**	82.91	95.06	89.03	93.18	82.94	89.20	87.28
DiaBlo	0.88	74.70	89.20	82.70	95.20	89.20	93.10	82.90	89.00	87.00
SWIFT	0.88	**76.30**	**90.32**	**83.31**	95.90	**89.27**	**93.52**	**84.18**	**91.60**	**88.05**

4.2 Datasets

We evaluate SWIFT across 17 datasets in three task categories. For commonsense reasoning, we fine-tune LLaMA3-8B [Grattafiori et al., 2024] on Commonsense-170K and evaluate on eight datasets [Hu et al., 2023]. For natural language generation, we separately fine-tune LLaMA2-7B [Touvron et al., 2023] on 100K examples each from MetaMathQA [Yu et al., 2023] and CodeFeedback [Zheng et al., 2024], using the data prepared by PiSSA [Meng et al., 2024], and evaluate on GSM8K, MATH, HumanEval, and MBPP. For image classification, we fine-tune CLIP ViT-B/32 [Radford et al., 2021] on five datasets [Fan et al., 2025]. Dataset descriptions, references, and experimental settings are provided in Appendices A.1 and A.2.

4.3 Comparative Results

We compare SWIFT with full fine-tuning, LoRA variants, and sparse or structured fine-tuning methods across the three task categories. In Tables 1–3, Avg. denotes the arithmetic mean of the

displayed task scores. Average-score differences reported below are calculated from the displayed averages.

4.3.1 Commonsense Reasoning

As shown in Table 1, SWIFT achieves the highest average accuracy of 88.05% at a reported trainable-parameter ratio of 0.88%. It outperforms LoRA by 3.04 percentage points and the second-best method, SpaRTA, by 0.77 percentage points. SWIFT also surpasses DiaBlo by 1.05 percentage points, with both methods reporting a trainable-parameter ratio of 0.88%. This comparison supports the benefit of combining block-diagonal adaptation with permutation-based routing.

4.3.2 Natural Language Generation

As shown in Table 2, SWIFT achieves the highest average score of 32.56% at a reported trainable-parameter ratio of 1.50%, surpassing full fine-tuning by 4.26 percentage points. It outperforms LoRA by 7.20 percentage points, the second-best method MoRe by 0.59 percentage points, and SpaRTA by 1.30 percentage points. SWIFT also exceeds DiaBlo by 1.03 percentage points, with both methods reporting a trainable-parameter ratio of 1.50%. SWIFT achieves the best results on GSM8K, MATH, and MBPP, demonstrating effective adaptation across mathematical reasoning and code generation.

Table 2: Performance comparison of LLaMA2-7B on four natural language generation datasets. Scores are percentages. Bold and underlined entries indicate the best and second-best scores, respectively, including ties.

Method	Param. (%)	GSM8K	MATH	HumanEval	MBPP	Avg.
Full Fine-Tuning	100	49.05	7.22	21.34	35.59	28.30
Low-rank adaptation						
LoRA	4.75	42.30	5.50	18.29	35.34	25.36
DoRA	4.76	50.42	7.26	23.78	<u>38.85</u>	30.08
PiSSA	4.75	53.07	7.44	21.95	37.09	29.89
HydraLoRA	1.87	49.58	7.02	23.17	37.34	29.28
Sparse and structured adaptation						
MoRe	2.37	<u>53.82</u>	<u>9.40</u>	<u>26.83</u>	37.84	<u>31.97</u>
SpaRTA	0.76	53.50	8.00	**27.44**	36.09	31.26
DiaBlo	1.50	53.07	9.22	26.22	37.59	31.53
SWIFT	1.50	**54.36**	**9.70**	<u>26.83</u>	**39.35**	**32.56**

4.3.3 Image Classification

To evaluate applicability beyond language tasks, we test SWIFT on five image classification datasets using CLIP ViT-B/32, as shown in Table 3. SWIFT achieves the highest average accuracy of 86.78%, outperforming full fine-tuning by 2.19 percentage points at a reported trainable-parameter ratio of 1.70%. It also surpasses LoRA by 8.09 percentage points, SpaRTA by 3.58 percentage points, and DiaBlo by 1.71 percentage points. SWIFT achieves the best accuracy on all five evaluated datasets, supporting the effectiveness of its update structure for visual adaptation.

4.4 Ablation Study

Update-density sensitivity. We study the sensitivity of SWIFT to the update density by varying the number of blocks N in the block-diagonal update. When the input and output dimensions are divisible by N, the structural density of each adapted matrix is $1/N$. As shown in Figure 3(a), densities of 0.20%, 0.39%, 0.78%, 1.56%, and 3.13% correspond to $N = 512, 256, 128, 64, 32$, respectively. SWIFT achieves its highest average accuracy of 88.05% at $N = 64$. Increasing the update density beyond this setting does not further improve accuracy in this experiment. These results support $N = 64$ as a practical default for the evaluated settings.

Table 3: Performance comparison of CLIP ViT-B/32 on five image classification datasets. Task scores are accuracies (%). Bold and underlined entries indicate the best and second-best scores, respectively.

Method	Param. (%)	Cars	DTD	GTSRB	RESISC45	SVHN	Avg.
Full Fine-Tuning	100	60.33	73.88	<u>98.30</u>	93.65	<u>96.78</u>	84.59
Low-rank adaptation							
LoRA	1.49	41.02	70.15	96.51	90.38	95.39	78.69
DoRA	1.49	40.75	71.91	97.71	90.19	95.46	79.20
PiSSA	1.49	40.41	69.62	95.84	90.58	95.84	78.46
GOAT	2.24	51.44	73.99	98.06	93.50	96.55	82.71
Sparse and structured adaptation							
MoRe	1.68	62.63	<u>74.26</u>	98.16	<u>93.68</u>	96.24	84.99
SpaRTA	1.60	58.40	70.40	97.90	92.90	96.40	83.20
DiaBlo	1.70	<u>63.67</u>	73.67	97.98	93.56	96.49	<u>85.07</u>
SWIFT	1.70	**67.69**	**76.06**	**98.57**	**94.46**	**97.10**	**86.78**

Permutation type. We further compare routing schemes in Figure 3(b), including diagonal cyclic (DC), anti-diagonal cyclic (AC), single-row swap (SS), random permutation (RP), and uniform scattering (US). US achieves the highest average accuracy of 86.78%, outperforming RP by 0.77 percentage points and DC by 1.71 percentage points. These results support the benefit of the deterministic interleaving construction, which distributes locally mixed outputs across feature groups.

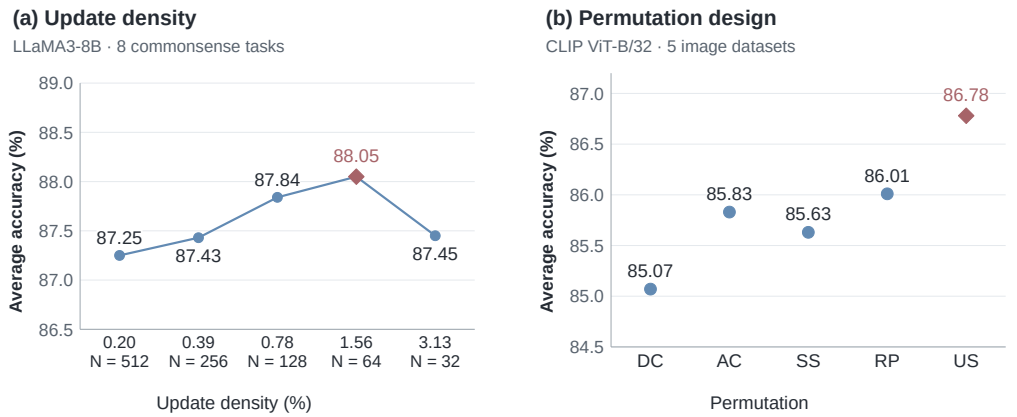


Figure 3: Ablation study of SWIFT. (a) Update-density sensitivity on eight commonsense reasoning datasets with LLaMA3-8B. Densities of 0.20%, 0.39%, 0.78%, 1.56%, and 3.13% correspond to $N = 512, 256, 128, 64, 32$, respectively. (b) Comparison of permutation routing schemes on five image classification datasets with CLIP ViT-B/32. DC, AC, SS, RP, and US denote diagonal cyclic, anti-diagonal cyclic, single-row swap, random permutation, and uniform scattering, respectively.

Additional analyses. On LLaMA3-8B, SWIFT improves over DiaBlo by 1.24, 0.93, and 1.05 percentage points for attention-only, feed-forward-only, and combined placement, respectively, with matched budgets within each placement. Across seeds 6, 24, and 42, SWIFT achieves 88.05% mean accuracy with a sample standard deviation of 0.10 percentage points. Appendix C provides the complete placement and seed results, together with the training-loss comparison.

4.5 Practical Efficiency

We conduct an additional efficiency comparison on LLaMA3-8B. All compared methods use a matched trainable-parameter budget of 0.88% and are trained on a single NVIDIA Tesla V100 GPU with 32 GB of memory using FP16 precision. The comparison uses the same data, splits, and

optimizer settings, with method-specific training durations: SWIFT is trained for two epochs and DiaBlo for three epochs. Results are averaged over seeds 6, 24, and 42. Memory cost denotes the maximum single-GPU training memory under the shared measurement protocol. Training time and FLOPs denote the total wall-clock time and cumulative floating-point operations over each method's complete training run.

As shown in Figure 4, under the respective training schedules, SWIFT has the shortest total training time and the lowest cumulative FLOP count among the compared implementations, reducing training time by 21.8–60.4% and total training FLOPs by 18.5–33.3% relative to the eight baselines. Its memory cost is 21,040 MB, only 16 MB above the minimum achieved by DiaBlo.

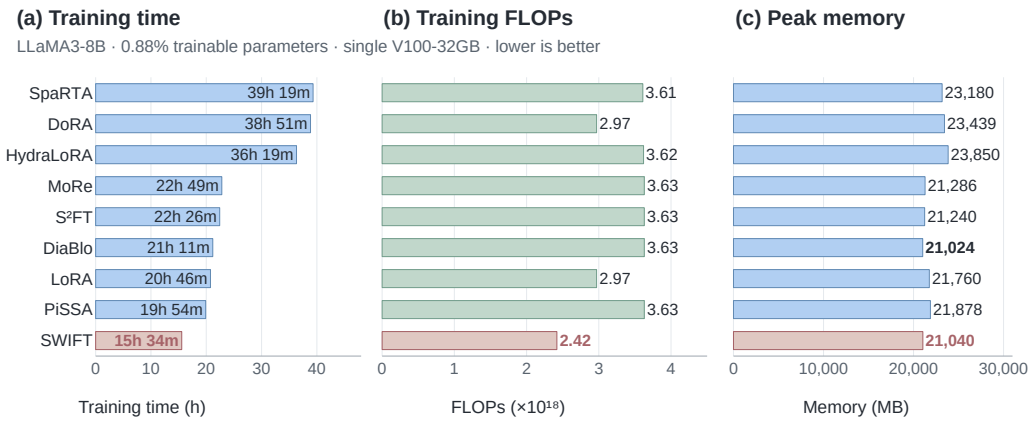


Figure 4: End-to-end efficiency comparison on LLaMA3-8B at a matched 0.88% trainable-parameter budget on one V100-32GB GPU. (a) Total training time. (b) Total training FLOPs. (c) Peak training memory. Training durations follow the method-specific schedules described in the text. Results are averaged over three seeds; lower values are better.

For merged-model inference, the effective weight is formed once as $\mathbf{W}_{\text{merged}} = \mathbf{W} + \mathbf{BP}$, after which the original dense linear layer is used. In the additional single-GPU LLaMA3-8B commonsense-reasoning inference comparison with batch size 1, SWIFT and the evaluated mergeable baselines achieve the same reported throughput of 6.83 samples/s.

5 Conclusion

We proposed SWIFT, a sparse fine-tuning framework inspired by small-world connectivity. By combining block-diagonal sparse updates with a fixed deterministic scattering permutation, SWIFT performs local feature mixing followed by cross-group routing under a low parameter budget. We further provided a topology-driven analysis showing that certain random sparse-update models and block-diagonal updates can exhibit information islands, whereas SWIFT guarantees positive block-level expansion under the stated structural and scattering conditions. Extensive experiments on commonsense reasoning, natural language generation, and image classification demonstrate that SWIFT achieves higher average performance than the evaluated low-rank and sparse fine-tuning baselines in each task category. Overall, SWIFT offers both an effective PEFT method and a graph-theoretic perspective for designing sparse update structures with improved cross-group connectivity.

6 Limitations

SWIFT achieves strong results across language and vision tasks through structured scattering. Its theoretical expansion guarantees require equal-sized, structurally dense square blocks and a sufficient scattering condition. Building on this foundation, exploring more flexible connectivity patterns while preserving parameter efficiency, and extending the theoretical guarantees to more general settings, are promising directions for future work.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62372132 and in part by the Major Key Project of Pengcheng Laboratory under Grant PCL2025A14. The authors would also like to thank Huawei Ascend Cloud Ecological Development Project for providing high-performance Ascend 910 processors. The authors declare no competing interests.

References

Jiayi Kuang, Ying Shen, Jingyou Xie, Haohao Luo, Zhe Xu, Ronghao Li, Yinghui Li, Xianfeng Cheng, Xika Lin, and Yu Han. Natural language understanding and inference with mllm in visual question answering: A survey. *ACM Computing Surveys*, 57(8):1–36, 2025.

Agnes Axelsson and Gabriel Skantze. Using large language models for zero-shot natural language generation from knowledge graphs. *arXiv preprint arXiv:2307.07312*, 2023.

Weikang Meng, Yadan Luo, Liangyu Huo, Yingjian Li, Yaowei Wang, Zheng Zhang, and Heng Tao Shen. Polaformer++: Polarity-aware linear attention with channel-wise spikiness. *IEEE Trans. Pattern Anal. Mach. Intell.*, 48(11):15412–15426, 2026. doi: 10.1109/TPAMI.2026.3723414. URL `https://doi.org/10.1109/TPAMI.2026.3723414`.

Weikang Meng, Yadan Luo, Liangyu Huo, Yaowei Wang, Xin Li, and Zheng Zhang. Nalaformer: Norm-aware linear attention for transformer models. *CoRR*, abs/2506.21137, 2025. doi: 10.48550/ARXIV.2506.21137. URL `https://doi.org/10.48550/arXiv.2506.21137`.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024a.

Cong Jiang, Xiaofeng Zhang, Fangzhi Zhu, XiaoWei Chen, Junxiong Zhu, and Zheng Zhang. Rethinking LLM reasoning: From explicit trajectories to latent representations. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=CbK7lYbmv8`.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, et al. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*, 2023a.

Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities. *arXiv preprint arXiv:2408.13296*, 2024.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024a.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Ziyu Zhao, Leilei Gan, Guoyin Wang, Wangchunshu Zhou, Hongxia Yang, Kun Kuang, and Fei Wu. Loraretriever: Input-aware lora retrieval and composition for mixed tasks in the wild. *arXiv preprint arXiv:2402.09997*, 2024b.

Yibo Yang, Xiaojie Li, Zhongzhu Zhou, Shuaiwen Song, Jianlong Wu, Liqiang Nie, and Bernard Ghanem. Corda: Context-oriented decomposition adaptation of large language models for task-aware parameter-efficient fine-tuning. *Advances in Neural Information Processing Systems*, 37: 71768–71791, 2024b.

Dan Zhang, Tao Feng, Lilong Xue, Yuandong Wang, Yuxiao Dong, and Jie Tang. Parameter-efficient fine-tuning for foundation models. *arXiv preprint arXiv:2501.13787*, 2025.

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. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023b.

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 *Forty-first International Conference on Machine Learning*, 2024.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37: 121038–121072, 2024.

Cong Jiang, Fangzhi Zhu, Xiaowei Chen, Junxiong Zhu, Bo Zheng, Yifeng Wang, and Zheng Zhang. Dmlora: Dynamic multi-subspace low-rank adaptation. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 1038–1042, 2025.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

Zhiwei He, Zhaopeng Tu, Xing Wang, Xingyu Chen, Zhijie Wang, Jiahao Xu, Tian Liang, Wenxiang Jiao, Zhuosheng Zhang, and Rui Wang. Rasa: Rank-sharing low-rank adaptation. *arXiv preprint arXiv:2503.12576*, 2025a.

Weixi Song, Zuchao Li, Lefei Zhang, Hai Zhao, and Bo Du. Sparse is enough in fine-tuning pre-trained large language models. *arXiv preprint arXiv:2312.11875*, 2023.

Xinyu Yang, Jixuan Leng, Geyang Guo, Jiawei Zhao, Ryumei Nakada, Linjun Zhang, Huaxiu Yao, and Beidi Chen. S^2ft: Efficient, scalable and generalizable llm fine-tuning by structured sparsity. *Advances in Neural Information Processing Systems*, 37:59912–59947, 2024c.

Luping Wang, Sheng Chen, Linnan Jiang, Shu Pan, Runze Cai, Sen Yang, and Fei Yang. Parameter-efficient fine-tuning in large language models: a survey of methodologies. *Artificial Intelligence Review*, 58(8):227, 2025.

Wenxuan Tan, Nicholas Roberts, Tzu-Heng Huang, Jitian Zhao, John Cooper, Samuel Guo, Chengyu Duan, and Frederic Sala. More fine-tuning with 10x fewer parameters. *arXiv preprint arXiv:2408.17383*, 2024.

Selcuk Gurses, Aozhong Zhang, Yanxia Deng, Xun Dong, Xin Li, Naigang Wang, Penghang Yin, and Zi Yang. Diablo: Diagonal blocks are sufficient for finetuning. *arXiv preprint arXiv:2506.03230*, 2025.

Chaitanya K Joshi. Transformers are graph neural networks. *arXiv preprint arXiv:2506.22084*, 2025.

Jesus Rios Aliaga, Pierre Dognin, Ronny Luss, and Karthikeyan Natesan Ramamurthy. Sparsity may be all you need: Sparse random parameter adaptation. In *Conference on Empirical Methods in Natural Language Processing*, 2025.

Ahmed Lachgar and Abdelfattah Achahbar. Uncovering the hidden structure of small-world networks. *Scientific Reports*, 14(1):6555, 2024.

João Pedro C Morais and Ruben Interian. A simple and flexible algorithm to generate real-world networks. In *International Conference on Complex Networks*, pages 1–12. Springer, 2025.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Cheng-Zhong Xu. Hydralora: An asymmetric lora architecture for efficient fine-tuning. *Advances in Neural Information Processing Systems*, 37: 9565–9584, 2024.

Chenghao Fan, Zhenyi Lu, Sichen Liu, Chengfeng Gu, Xiaoye Qu, Wei Wei, and Yu Cheng. Make lora great again: Boosting lora with adaptive singular values and mixture-of-experts optimization alignment. *arXiv preprint arXiv:2502.16894*, 2025.

Yuchen Zeng and Kangwook Lee. The expressive power of low-rank adaptation. *arXiv preprint arXiv:2310.17513*, 2023. URL `https://arxiv.org/abs/2310.17513`.

Haoze He, Juncheng B Li, Xuan Jiang, and Heather Miller. Smt: Fine-tuning large language models with sparse matrices. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Alan Ansell, Ivan Vulić, Hannah Sterz, Anna Korhonen, and Edoardo M Ponti. Scaling sparse fine-tuning to large language models. *arXiv preprint arXiv:2401.16405*, 2024.

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.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*, 2023.

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.

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.

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.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 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. *arXiv preprint arXiv:1905.10044*, 2019.

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*, volume 34, pages 7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

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.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 554–561, 2013.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3606–3613, 2014.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *The 2011 international joint conference on neural networks*, pages 1453–1460. IEEE, 2011.

Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, 2017.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. URL `https://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf`.

A Datasets and Experimental Settings

A.1 Datasets

Commonsense Reasoning Tasks. We evaluate the commonsense reasoning capabilities of SWIFT on the following datasets:

1. **BoolQ** Clark et al. [2019]: A question-answering dataset comprising yes/no questions.
2. **PIQA** (Physical Interaction Question Answering) Bisk et al. [2020]: A dataset of questions accompanied by two candidate solutions, where identifying the correct answer requires physical commonsense reasoning.
3. **SIQA** (Social IQA) Sap et al. [2019]: A question-answering dataset focused on reasoning about motivations, reactions, and social implications in everyday situations.
4. **HellaSwag** Zellers et al. [2019]: A commonsense sentence-completion benchmark in which each context is accompanied by multiple candidate endings.
5. **WinoGrande** Sakaguchi et al. [2021]: A fill-in-the-blank benchmark with two candidate options for each sentence.
6. **ARC-e/ARC-c** (AI2 Reasoning Challenge–Easy/Challenge) Clark et al. [2018]: The Easy and Challenge subsets of ARC, containing multiple-choice questions about school-level science.
7. **OBQA** (OpenBookQA) Mihaylov et al. [2018]: A question-answering dataset requiring reasoning that combines scientific facts with additional common knowledge.

Following Hu et al. [2023], we use Commonsense-170K for training. This dataset contains approximately 170,000 examples in a standardized instruction format, covering diverse aspects of commonsense reasoning.

Natural Language Generation Tasks. We evaluate the natural language generation capabilities of SWIFT on the following datasets:

1. **GSM8K** Cobbe et al. [2021]: A mathematical reasoning dataset comprising high-quality, linguistically diverse grade-school mathematical word problems written by human problem creators.
2. **MATH** Hendrycks et al. [2021]: A mathematical reasoning dataset comprising competition mathematics problems across seven subject categories.
3. **HumanEval** Chen et al. [2021]: A code generation benchmark that evaluates the functional correctness of generated Python programs using unit tests.
4. **MBPP** Austin et al. [2021]: A benchmark of crowdsourced Python programming problems intended to be approachable for novice programmers, covering basic programming concepts and standard library functionality.

Following Meng et al. [2024], we use the following two datasets for training:

1. **Math: MetaMathQA** Yu et al. [2023]. MetaMathQA is a synthetic dataset developed to improve mathematical reasoning in large language models. We use 100K examples from the MetaMathQA training data prepared by PiSSA and evaluate the resulting model on GSM8K and MATH.
2. **Code: Code-Feedback** Zheng et al. [2024]. We use 100K examples from the Python instruction-response data prepared by PiSSA from CodeFeedback and evaluate the resulting model on HumanEval and MBPP.

For both training datasets, the released scripts select the first 100K examples from the corresponding training split.

Image Classification Tasks. We evaluate SWIFT on the following image classification datasets:

1. **Cars (Stanford Cars)** Krause et al. [2013]: A car classification dataset containing 16,185 images across 196 classes, with separate training and test sets.

2. **DTD (Describable Textures Dataset)** Cimpoi et al. [2014]: A texture classification dataset containing 5,640 images across 47 classes, with 120 images per class.
3. **GTSRB (German Traffic Sign Recognition Benchmark)** Stallkamp et al. [2011]: A traffic sign classification dataset containing over 50,000 images across 43 categories.
4. **RESISC45** Cheng et al. [2017]: A remote sensing scene classification dataset containing 31,500 images across 45 categories, with 700 images per category.
5. **SVHN (Street View House Numbers)** Netzer et al. [2011]: A digit classification dataset derived from Google Street View images, containing 10 classes, with 73,257 training images, 26,032 test images, and an additional set of 531,131 images. Our implementation uses the official training and test splits without the additional set.

Following Fan et al. [2025], we report classification accuracy on all five datasets.

A.2 Experimental Settings

Implementation Details. SWIFT is implemented in PyTorch using Hugging Face Transformers. We use LLaMA3-8B for commonsense reasoning, LLaMA2-7B for natural language generation, and CLIP ViT-B/32 for image classification. The released SWIFT training scripts default to one GPU with FP16 precision for commonsense reasoning, two GPUs with BF16 precision for natural language generation, and four GPUs with BF16 mixed precision for image classification. The distributed generation and image-classification configurations use DeepSpeed ZeRO Stage 2.

The pre-trained backbone weights remain frozen during SWIFT fine-tuning. The trainable block-diagonal updates are initialized to zero, and the scattering permutation is fixed throughout training. For image classification, the classification head is trained jointly with the SWIFT updates.

All downstream experiments, including the main comparisons, ablations, and efficiency evaluations, use random seeds 6, 24, and 42, and the reported results are averaged across the three runs.

Hyperparameters. For SWIFT, the number of blocks is set to 64 in all main experiments; the block-number ablation varies this setting explicitly. We use the same deterministic scattering construction across tasks without task-specific permutation search. SWIFT uses AdamW with zero weight decay and does not apply dropout within its update branch. Detailed task-specific settings are provided below.

For commonsense reasoning, full fine-tuning uses a learning rate of 1×10^{-5}. LoRA, DoRA, PiSSA, and HydraLoRA use a learning rate of 3×10^{-4} with rank 32. SpIEL and SMT follow the hyperparameter settings reported in the SMT paper, and S^2FT follows the settings reported in the S^2FT paper. SpaRTA, DiaBlo, MoRe, and SWIFT use a learning rate of 5×10^{-5}.

SWIFT adapts the query, key, and value projections and the feed-forward up and down projections, corresponding to `q_proj`, `k_proj`, `v_proj`, `up_proj`, and `down_proj`. Training lasts two epochs with a global batch size of 16, a per-device micro-batch size of 2, and eight gradient accumulation steps in the default single-GPU configuration. The maximum sequence length is 256 tokens, and the learning-rate schedule uses 100 warmup steps. Prompt tokens are excluded from the training loss. We hold out 120 examples from the training data for validation using split seed 42. Validation and checkpoint saving occur every 80 optimizer steps, and the checkpoint with the lowest validation loss is selected.

For natural language generation, full fine-tuning follows the hyperparameter settings used in the PiSSA paper. LoRA, DoRA, and PiSSA use a learning rate of 2×10^{-5} with rank 160, following PiSSA. HydraLoRA uses a learning rate of 2×10^{-5} with rank 32. SpaRTA, DiaBlo, MoRe, and SWIFT use a learning rate of 2×10^{-4}.

SWIFT adapts all attention projections and the feed-forward gate, up, and down projections, corresponding to `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`. Math and code models are fine-tuned separately for one epoch. The maximum sequence length is 512 tokens. The default configuration uses a per-device micro-batch size of 1 and two gradient accumulation steps on two GPUs, giving a global batch size of 4. We use a cosine learning-rate schedule with a warmup ratio of 0.03 and evaluate the final checkpoint.

For image classification, full fine-tuning, LoRA, DoRA, and PiSSA follow the hyperparameter settings used in the GOAT paper, with a learning rate of 1×10^{-4} and rank 8 for the low-rank methods. SpaRTA, DiaBlo, and MoRe use a learning rate of 1×10^{-3}. SWIFT uses a learning rate of 1×10^{-4}.

SWIFT adapts the linear layers of the vision backbone, including the attention and feed-forward projections. The classification head is trained directly. Training lasts 35, 76, 11, 15, and 4 epochs for Cars, DTD, GTSRB, RESISC45, and SVHN, respectively. The default configuration uses a global batch size of 512, a per-device micro-batch size of 64, and two gradient accumulation steps on four GPUs. We use a cosine learning-rate schedule with a warmup ratio of 0.03.

The vision data loader reserves 10% of each supplied training split, capped at 5,000 examples, using split seed 0, and uses the remaining examples for optimization. Training images undergo random resized cropping to 224×224 pixels, with a crop scale of $[0.9, 1.0]$, followed by CLIP normalization. Evaluation images are resized and center-cropped to 224×224 pixels using the same normalization. Evaluation accuracy is recorded at the end of each epoch, and the final adapter and classification head are saved after training. All SWIFT training scripts complete the specified epoch schedule without early stopping.

Evaluation Settings. For commonsense reasoning, the released evaluation script uses a batch size of 1, four-beam decoding, and a maximum of 32 generated tokens. Predictions are extracted using the task-specific answer formats and evaluated by accuracy. For natural language generation, the math and code evaluation scripts use greedy decoding with a batch size of 4 and a maximum of 1,024 generated tokens. Mathematical answers are extracted and compared with the references using the provided evaluation utilities. Code generation is evaluated with EvalPlus using one generated completion per problem.

B Proof of Small-World Induced Fine-Tuning Theory

This appendix proves Theorems 3.3, 3.4, and 3.6, together with the MRGC bound in Equation (15) and the coordinate-level expansion bound in Equation (16). Throughout, graphs are induced by the structural support of trainable updates, including trainable entries initialized to zero. Under the row-vector convention, an edge (i, j) represents a possible direct trainable interaction from input coordinate i to output coordinate j. We use

$$O(S) = \{j \in V : \exists i \in S,\ (i, j) \in E\}, \qquad \partial_{\mathrm{out}} S = O(S) \setminus S.$$

The quantities $h(G)$ and $h_{\mathcal{P}}(G)$, defined in Equations (9) and (13), measure directed out-vertex expansion over coordinate subsets and block unions, respectively.

B.1 Additional Coverage and Expansion Bounds

The inverse permutation satisfies the same scattering bound, since

$$|\pi^{-1}(V_r) \cap V_q| = |\pi(V_q) \cap V_r| \le \epsilon m.$$

Consequently, this condition controls the output routing under the convention in Equation 4.

For a union U_I of s blocks, structural density gives $O(U_I) = \pi^{-1}(U_I)$ and $|O(U_I)| = sm$. The scattering condition bounds its overlap with U_I by $s^2 \epsilon m$, yielding Equation 14. The same condition also provides a separate remote-group coverage bound:

$$\mathrm{MRGC} \ge \frac{\lceil 1/\epsilon \rceil - 1}{k - 1}. \tag{15}$$

Indeed, every source coordinate has m output neighbors, with at most ϵm in any target block, so it reaches at least $\lceil 1/\epsilon \rceil$ blocks, at most one of which is local.

Coordinate-level expansion. Under the assumptions of Theorem 3.6, a separate coordinate-level bound holds:

$$h(G_{\mathbf{BP}}) \ge \min\left\{1 - \frac{k\epsilon}{2},\ c(k)\right\} > 0, \qquad c(k) = \begin{cases} 2/k, & k \text{ even}, \\ 1/k, & k \text{ odd}. \end{cases} \tag{16}$$

This follows by considering how many blocks a coordinate subset intersects and using the structurally dense support within each block. The block-level theorem, MRGC bound, and coordinate-level bound are derived separately from the same scattering condition. Detailed proofs are provided in Appendix B.4.

Under exact uniform scattering, $\epsilon = 1/k$, Theorem 3.6 gives $h_{\mathcal{P}}(G_{\mathbf{BP}}) \geq 1/2$. For other compatible dimensions, the realized value in Equation 5 determines whether the sufficient condition $\epsilon_{\text{real}} < 2/k$ holds. Permutations that only exchange entire blocks have $\epsilon = 1$; in particular, swapping disjoint block pairs leaves closed block unions when $k \geq 4$. The expansion guarantees above apply to equal-sized, structurally dense square blocks, while rectangular projections are characterized through their bipartite group coverage.

B.2 Proof of Theorem 3.3: Connectivity Collapse under Independent Random Sparsification

Let G_d be the interaction graph induced by the random structural support, and let $D_{\text{out}}(v)$ denote the out-degree of vertex v. Write

$$p_{v,d} = \mathbb{P}\big(D_{\text{out}}(v) = 0\big).$$

The uniform lower-bound assumption

$$\liminf_{d\to\infty} \min_{v\in\{1,\dots,d\}} p_{v,d} > 0$$

implies that there exist constants $\eta \in (0,1)$ and d_0 such that

$$p_{v,d} \geq \eta \qquad \text{for every } d \geq d_0 \text{ and every } v \in \{1,\dots,d\}. \tag{17}$$

Let

$$X_d = \sum_{v=1}^{d} \mathbb{I}_{\{D_{\text{out}}(v)=0\}}$$

be the number of zero-out-degree vertices. Since outgoing supports are sampled independently across vertices, the events $\{D_{\text{out}}(v) = 0\}$ are independent. Therefore, for $d \geq d_0$,

$$\mathbb{P}(X_d = 0) = \prod_{v=1}^{d} (1 - p_{v,d}) \leq (1-\eta)^d \xrightarrow[d\to\infty]{} 0. \tag{18}$$

Thus, $\mathbb{P}(X_d \geq 1) \to 1$.

For $d \geq 2$, if a vertex v has zero out-degree, then the singleton $S = \{v\}$ is admissible in Equation (9) and satisfies

$$\partial_{\text{out}} S = \emptyset.$$

Consequently,

$$0 \leq h(G_d) \leq \frac{|\partial_{\text{out}} S|}{|S|} = 0.$$

It follows that

$$\mathbb{P}\big(h(G_d) = 0\big) \geq \mathbb{P}(X_d \geq 1) \geq 1 - (1-\eta)^d \xrightarrow[d\to\infty]{} 1. \tag{19}$$

This proves Theorem 3.3. ■

For the independent Bernoulli model with entry probability a/d, the expected out-degree is a and

$$p_{v,d} = \Big(1 - \frac{a}{d}\Big)^d \xrightarrow[d\to\infty]{} e^{-a} > 0. \tag{20}$$

Hence this model satisfies the theorem's assumptions. The proof applies more generally whenever outgoing supports are independent across vertices and the zero-out-degree probabilities satisfy the stated uniform lower bound.

B.3 Proof of Theorem 3.4: Zero Expansion of Block-Diagonal Updates

Let $\mathbf{B} \in \mathbb{R}^{d\times d}$ have block-diagonal structural support under the partition

$$V = \bigsqcup_{r=1}^{k} V_r, \qquad |V_r| = m, \qquad d = km, \qquad k > 1.$$

Thus,

$$\Omega_{\mathbf{B}} \subseteq \bigcup_{r=1}^{k} (V_r \times V_r).$$

Consider $S = V_1$. Since $k > 1$,

$$1 \le |S| = m \le \lfloor d/2 \rfloor.$$

For any source vertex $i \in S$, an edge $(i, j) \in \Omega_{\mathbf{B}}$ requires j to belong to the same block as i. Therefore $O(S) \subseteq S$, giving

$$\partial_{\text{out}} S = O(S) \setminus S = \emptyset.$$

Substituting this admissible set into Equation (9) yields

$$0 \le h(G_{\mathbf{B}}) \le \frac{|\partial_{\text{out}} S|}{|S|} = 0.$$

Hence $h(G_{\mathbf{B}}) = 0$, proving Theorem 3.4. ■

B.4 Proof of Theorem 3.6: Expansion of Small-World Induced Fine-Tuning

Let $\mathcal{P} = \{V_r\}_{r=1}^{k}$ be a partition of V into $k > 1$ blocks of equal size $m = d/k$. The trainable block-diagonal factor has structurally dense blocks:

$$\Omega_{\mathbf{B}} = \bigcup_{r=1}^{k} (V_r \times V_r).$$

Let $\mathbf{P}$ be the permutation matrix associated with π under the convention

$$(\mathbf{BP})_{ij} = \mathbf{B}_{i,\pi(j)}, \tag{21}$$

and write

$$\Delta\mathbf{W} = \mathbf{BP}, \qquad G = G_{\mathbf{BP}}.$$

Output neighborhoods and inverse scattering. For any source coordinate $i \in V_r$,

$$(i, j) \in \Omega_{\Delta} \iff (i, \pi(j)) \in \Omega_{\mathbf{B}} \iff \pi(j) \in V_r.$$

Structural density therefore gives the exact neighborhood identity

$$O(\{i\}) = \pi^{-1}(V_r). \tag{22}$$

In particular, every source coordinate has exactly m neighbors in the structural support.

The scattering assumption also holds for π^{-1}. Indeed, for all $r, q \in \{1, \dots, k\}$,

$$|\pi^{-1}(V_r) \cap V_q| = |V_r \cap \pi(V_q)| \le \epsilon m. \tag{23}$$

Block-level expansion. Consider any admissible block union

$$U_I = \bigcup_{r\in I} V_r, \qquad 1 \le |I| = s \le \lfloor k/2 \rfloor.$$

By Equation (22),

$$O(U_I) = \bigcup_{r\in I} \pi^{-1}(V_r) = \pi^{-1}(U_I). \tag{24}$$

Since π is a bijection,

$$|O(U_I)| = |U_I| = sm. \tag{25}$$

The sets $\{\pi^{-1}(V_r)\}_{r=1}^k$ form a partition of V, as do the sets $\{V_q\}_{q=1}^k$. Consequently, the following intersections are pairwise disjoint, and Equation (23) gives

$$\begin{aligned} |O(U_I) \cap U_I| &= |\pi^{-1}(U_I) \cap U_I| \\ &= \sum_{r \in I} \sum_{q \in I} |\pi^{-1}(V_r) \cap V_q| \\ &\leq s^2 \epsilon m. \end{aligned} \tag{26}$$

Combining Equations (25) and (26),

$$\begin{aligned} |\partial_{\text{out}} U_I| &= |O(U_I)| - |O(U_I) \cap U_I| \\ &\geq sm - s^2 \epsilon m \\ &= sm(1 - s\epsilon). \end{aligned} \tag{27}$$

Dividing by $|U_I| = sm$ yields

$$\frac{|\partial_{\text{out}} U_I|}{|U_I|} \geq 1 - s\epsilon \geq 1 - \frac{k\epsilon}{2}. \tag{28}$$

Taking the minimum over all admissible block unions gives

$$h_{\mathcal{P}}(G) \geq 1 - \frac{k\epsilon}{2} = 1 - \frac{d\epsilon}{2m}. \tag{29}$$

Under the sufficient condition $\epsilon < 2/k$, this lower bound is strictly positive. This proves Theorem 3.6. ■

Proof of the MRGC bound. We next prove Equation (15) using the same partition, so that $K = k$ in the MRGC definition. Fix a source coordinate $i \in V_r$, and let

$$\mathcal{L}(i) = \{q \in \{1, \ldots, k\} : O(\{i\}) \cap V_q \neq \emptyset\}$$

be the set of all target groups reached by i, including its local group if reached. By Equations (22) and (23),

$$|O(\{i\})| = m, \qquad |O(\{i\}) \cap V_q| \leq \epsilon m \quad \text{for every } q.$$

Therefore,

$$m = \sum_{q \in \mathcal{L}(i)} |O(\{i\}) \cap V_q| \leq |\mathcal{L}(i)|\, \epsilon m,$$

which implies

$$|\mathcal{L}(i)| \geq \lceil 1/\epsilon \rceil.$$

Excluding the local group removes at most one group, so

$$|\mathcal{R}(i)| \geq \lceil 1/\epsilon \rceil - 1.$$

Since this holds for every source coordinate,

$$\text{MRGC} = \min_{i \in V} \frac{|\mathcal{R}(i)|}{k-1} \geq \frac{\lceil 1/\epsilon \rceil - 1}{k-1}. \tag{30}$$

This proves Equation (15). ■

Proof of the coordinate-level expansion bound. We finally prove Equation (16). Let $S \subseteq V$ be any set satisfying

$$1 \leq |S| \leq \lfloor d/2 \rfloor.$$

Define

$$I_S = \{r : S \cap V_r \neq \emptyset\}, \qquad t = |I_S|, \qquad U_{I_S} = \bigcup_{r \in I_S} V_r.$$

Every coordinate in a given source block has the same output neighborhood by Equation (22). Since S intersects every block indexed by I_S,

$$O(S) = O(U_{I_S}) = \pi^{-1}(U_{I_S}), \qquad |O(S)| = tm. \tag{31}$$

We consider two cases.

Case 1: $t \le k/2$. Since $S \subseteq U_{I_S}$, the overlap estimate in Equation (26), applied to U_{I_S}, gives

$$|O(S) \cap S| \le |\pi^{-1}(U_{I_S}) \cap U_{I_S}| \le t^2 \epsilon m.$$

Consequently,

$$|\partial_{\text{out}} S| = |O(S)| - |O(S) \cap S| \ge tm(1 - t\epsilon).$$

The assumption $\epsilon < 2/k$ implies

$$1 - t\epsilon \ge 1 - \frac{k\epsilon}{2} > 0.$$

Using $|S| \le tm$, we obtain

$$\frac{|\partial_{\text{out}} S|}{|S|} \ge \frac{tm(1 - t\epsilon)}{|S|} \ge 1 - t\epsilon \ge 1 - \frac{k\epsilon}{2}. \tag{32}$$

Case 2: $t > k/2$. Using Equation (31) and the bound $|O(S) \cap S| \le |S|$, we have

$$|\partial_{\text{out}} S| \ge tm - |S|.$$

Since $|S| \le d/2 = km/2$,

$$\frac{|\partial_{\text{out}} S|}{|S|} \ge \frac{tm}{|S|} - 1 \ge \frac{2t}{k} - 1. \tag{33}$$

The integer t satisfies $t \ge \lfloor k/2 \rfloor + 1$, and hence

$$\frac{2t}{k} - 1 \ge c(k), \qquad c(k) = \begin{cases} 2/k, & k \text{ even}, \\ 1/k, & k \text{ odd}. \end{cases}$$

Combining the two cases, every admissible coordinate subset S satisfies

$$\frac{|\partial_{\text{out}} S|}{|S|} \ge \min\left\{1 - \frac{k\epsilon}{2},\ c(k)\right\}.$$

Taking the minimum over S gives

$$h(G_{\mathbf{BP}}) \ge \min\left\{1 - \frac{k\epsilon}{2},\ c(k)\right\} > 0. \tag{34}$$

This proves Equation (16). ■

C Additional Analyses

C.1 Baseline Descriptions

To assess the effectiveness of SWIFT, we perform a comprehensive evaluation against several baseline methods, grouped into three categories:

1. **Full Fine-Tuning:** This baseline updates all pre-trained parameters, providing a reference for adaptation without restricting the trainable parameter budget.
2. **Low-Rank Adaptation Family:** This group includes LoRA [Hu et al., 2022] and its variants. We include DoRA [Liu et al., 2024] for weight-decomposed adaptation, PiSSA [Meng et al., 2024] for principal-component initialization, GOAT [Fan et al., 2025] for SVD-structured mixture-of-experts adaptation and optimization alignment, and HydraLoRA [Tian et al., 2024] for asymmetric adaptation to heterogeneous data. These methods achieve parameter efficiency through low-rank parameterizations and their extensions.
3. **Sparse and Structured Fine-Tuning Approaches:** This group includes random, dynamic, and structured tuning methods. We compare SWIFT against SpaRTA [Aliaga et al., 2025], which updates randomly selected parameters, and S^2FT [Yang et al., 2024c], which selects attention heads and feed-forward channels for structured sparse updates. We also include SMT [He et al., 2025b], which selects task-relevant submatrices using saliency signals, and SpIEL [Ansell et al., 2024], which dynamically reallocates sparse trainable parameters. Finally, we compare with MoRe [Tan et al., 2024], which uses Monarch-style structured parameterizations with fixed permutations, and DiaBlo [Gurses et al., 2025], which fine-tunes diagonal blocks of selected weight matrices.

C.2 Support Simulation and Rectangular Coverage

For rectangular layers, we use separate input and output vertex sets with N paired input and output groups. The remote target groups for an input coordinate are the output groups other than its paired group, and the normalization is $N - 1$. The square-graph expansion results in Section 3.3 apply to square layers.

We use MRGC to compare representative structural supports under a matched sparse budget, with full fine-tuning as a dense reference. We simulate $d = 4096$ feature dimensions partitioned into $K = 64$ equal-sized groups. The random sparse baseline uses the same expected density as block-diagonal tuning, $p = 1/64$, and results are averaged over $1{,}000$ sampled masks. Full fine-tuning uses a dense support, block-diagonal tuning retains only within-group interactions, and SWIFT uses the support of $\mathbf{BP}$ with the deterministic interleaving permutation.

Full fine-tuning achieves the maximal MRGC of 1.00, while block-diagonal tuning yields 0.00 due to the absence of cross-group interactions. Random sparse tuning improves coverage but remains uneven, achieving a mean MRGC of 0.41. In this setting, SWIFT attains an MRGC of 1.00 at density $1/64$. Thus, the scattering permutation provides complete remote-group coverage in this configuration without increasing the trainable parameter budget.

C.3 Placement and Seed Analyses

Attention and feed-forward placement. We additionally compare SWIFT and DiaBlo when updates are applied to attention projections, feed-forward projections, or both. Within each placement, the two methods use identical target modules and matched trainable-parameter budgets. Results are averaged over the eight commonsense reasoning datasets and seeds 6, 24, and 42. As shown in Figure 5(a), SWIFT improves average accuracy by 1.24, 0.93, and 1.05 percentage points, respectively. The improvement across all three placements supports the applicability of permutation-based routing to both module families.

Variation across random seeds. We additionally evaluate SWIFT on LLaMA3-8B using seeds 6, 24, and 42. The average accuracies across the eight commonsense reasoning datasets are 87.94%, 88.10%, and 88.11%, respectively, as shown in Figure 5(b). The mean accuracy is 88.05%, with a sample standard deviation of approximately 0.10 percentage points, computed across the three seed-level averages using the $n - 1$ denominator with $n = 3$. These statistics describe variation across random seeds in this setting. No additional statistical significance test was performed.

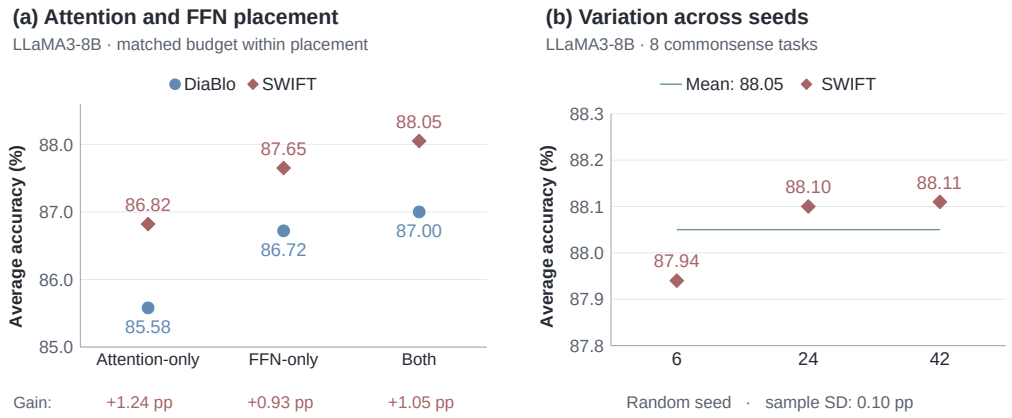


Figure 5: Placement and seed analyses on LLaMA3-8B. (a) SWIFT and DiaBlo use identical target modules and matched parameter budgets within each placement; scores are averaged over eight commonsense reasoning datasets and three seeds. (b) SWIFT accuracy for seeds 6, 24, and 42, with the horizontal line showing their mean.

C.4 Convergence Analysis

As shown in Figure 6, we compare the training loss curves of SWIFT, LoRA, and SpaRTA over two epochs. SWIFT reaches a lower final training loss than both baselines and maintains a low loss during the later stage of training. These curves support favorable optimization behavior under the evaluated training schedule. Wall-clock training efficiency is assessed separately in Section 4.5.

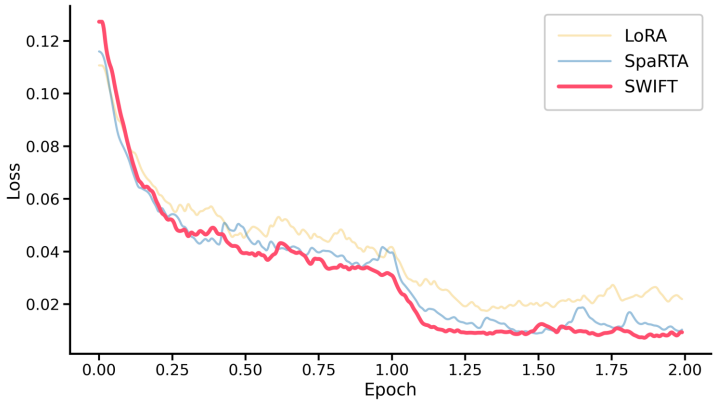


Figure 6: Training loss curves for LoRA, SpaRTA, and SWIFT when fine-tuning LLaMA3-8B on Commonsense-170K for commonsense reasoning.

D Broader Impacts and Asset Information

D.1 Broader Impacts

SWIFT reduces the number of trainable parameters required for adapting pre-trained models and demonstrates lower training costs in the evaluated efficiency comparison. These benefits may make task-specific adaptation more accessible to researchers with limited computational resources. Adapted models may inherit biases, factual inaccuracies, or other undesirable behaviors from their pre-trained backbones and training data. Parameter-efficient adaptation alone does not address these issues. Downstream deployment therefore requires application-specific evaluation and safeguards appropriate to the intended use.

D.2 Asset Attribution and Licenses

The original model and dataset publications are cited in the main text and Appendix A.1. LLaMA3-8B is distributed under the Meta Llama 3 Community License Agreement, and LLaMA2-7B under the Llama 2 Community License Agreement. The official CLIP implementation uses the MIT License, and PEFT uses the Apache License 2.0.

The SWIFT repository retains upstream license files and third-party notices, including the MIT license for the GOAT-derived image-classification code. Applicable subdirectory and file-level notices remain in effect. Pre-trained model weights and datasets are obtained separately from their original providers under the corresponding access conditions and terms of use; they are not redistributed with the SWIFT code.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have specifically summarized the main contributions of this article in the abstract and introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6 discusses the structural assumptions underlying the theoretical expansion guarantees and identifies extensions to more flexible connectivity patterns and more general settings as directions for future work.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We provide a detailed explanation and present a rigorous mathematical proof.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides comprehensive details on the dataset, hyperparameters and training procedures, ensuring that all necessary information is disclosed to fully reproduce the main experimental results and validate the claims and conclusions.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The SWIFT implementation is publicly available at `https://github.com/Zijun-Xiong/SWIFT`, including training and evaluation scripts, environment specifications, and data-preparation instructions for commonsense reasoning, natural language generation, and image classification. Pre-trained weights and datasets must be obtained separately from their original sources under the applicable access conditions. The release covers the SWIFT pipelines for these three task groups; it does not include dedicated reproduction scripts for all baseline comparisons, ablations, and efficiency measurements.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We made every effort to provide the necessary experimental details, including datasets and experimental settings.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: All downstream experiments, including the main comparisons, ablations, and efficiency evaluations, are conducted with random seeds 6, 24, and 42, and the reported results are averaged across these three runs. Appendix C.3 additionally provides the individual seed results for SWIFT on LLaMA3-8B commonsense reasoning, with a mean accuracy of 88.05% and a sample standard deviation of 0.10 percentage points across seed-level averages. No additional statistical significance tests were performed.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper specifies the type of compute resources used and provides details on memory usage and execution time, ensuring sufficient information is available to reproduce the experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research adheres to the NeurIPS Code of Ethics in all respects, ensuring ethical considerations are met in dataset, experimentation, and reporting of results.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Appendix D.1 discusses the potential benefits of making model adaptation more computationally accessible, the possibility of inheriting biases and factual inaccuracies from pre-trained models and training data, and the need for application-specific evaluation and safeguards.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper poses no such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Original model and dataset publications are cited, and Appendix D.2 identifies the licenses of the backbone models and principal software components. The released repository retains upstream license files and third-party notices.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The released SWIFT code includes documentation of the implementation, environment requirements, data preparation, and training and evaluation commands for the three task groups.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing, nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing, nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.